uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería en Tecnologías de Telecomunicaciones

2018-2019

Trabajo Fin de Grado

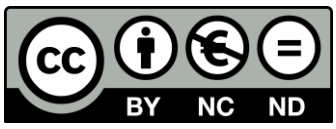
"EVALUACIÓN DEL USO DEL PROTOCOLO QUIC EN INTERNET"

Sergio Lázaro Espinosa

Tutor

Ignacio Soto Campos

Madrid, Julio 2019



[Incluir en el caso del interés de su publicación en el archivo abierto]

ÍNICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

AGRADECIMIENTOS

Ahora que se acerca el final de mi etapa como estudiante es inevitable pararse a pensar en estos años de carrera y en todas las personas que me han apoyado estos años, tanto los que ya estaban, como los que han llegado nuevos.

Me gustaría agradecer a mis padres, José Manuel y María Esther, el esfuerzo que han tenido que hacer y el apoyo que me han dado en todos estos años de estudio. También agradecer a mi hermana, María, por apoyarme siempre en mis decisiones y hacer que estudiar sea más fácil.

A mis compañeros de la Universidad Carlos III, por todas las horas y momentos que hemos compartido en clase, en la biblioteca, haciendo prácticas, proyectos…, porque gracias a ello he podido conocer más a esas personas y entablar una amistad duradera que va más allá del período lectivo.

A toda la gente que conocí durante mi erasmus en Kaiserslautern, porque hicisteis posible que, a pesar de estar a más de 2000 km de mi hogar, mi familia y mis amigos, me sintiera como en casa. Sobre todo, mencionar a Curro, Gerardo, Juanjo, Alkan, Davide, Gabor y María Beck-Friis, no sólo por hacer más amenas las largas tardes de estudio, sino por mostrarme la cultura de otros países, por acompañarme en viajes inolvidables y por vivir esta experiencia conmigo.

También a mi familia y amigos, que me han ayudado a ser constante y a no rendirme.

Por último, me gustaría agradecer a mi tutor Ignacio todas las horas dedicadas, ya que se ha volcado en mi trabajo y me ha ayudado con él incluso en vacaciones o fines de semana.

Gracias a todos.

RESUMEN

En este documento se evalúa el uso del protocolo QUIC utilizando como referencia algunos de los servidores más usados de Internet. Para exponer la información de una manera clara, primero se realiza un estudio del arte en el que se analiza y se explica brevemente el funcionamiento de los protocolos más importantes que gestionan el tráfico web en los últimos años (TCP, TLS, HTTP…). Tras este estudio, se puede ver que hay distintas limitaciones de prestaciones cuando se usan estos protocolos para servir tráfico web. Es en este contexto en el que surge el protocolo QUIC, para solucionar estas limitaciones y mejorar la transferencia de contenidos en paralelo y la latencia en las comunicaciones.

QUIC es un proyecto originado por la empresa Google y ya utilizado por dicha empresa. Sin embargo, hay otra versión que está siendo estandarizada por el IETF. Aunque son muy similares en cuanto a funcionamiento, tienen algunas diferencias que se explicarán en este trabajo. Además, se describe el diseño del protocolo y sus distintos mecanismos de control de flujo, control de congestión, seguridad… Una vez entendidos estos conceptos, el lector ya tiene una buena idea del funcionamiento del protocolo y puede entender los problemas que QUIC soluciona y las mejoras que aporta al tráfico en Internet.

Por último, de forma práctica se evalúa el uso de este protocolo en los servidores web más utilizados y se llevan a cabo diferentes medidas y comprobaciones mediante el analizador de tráfico Wireshark. También se describe brevemente el uso de QUIC en los navegadores web dominantes.

1. INTRODUCCIÓN

1.1 MOTIVACIÓN DEL TRABAJO

La motivación para realizar este trabajo surge de los conocimientos y habilidades adquiridas a lo largo del grado en el ámbito de las redes y los servicios de comunicaciones.

A esto se une el incremento del uso de las nuevas tecnologías y una demanda cada vez mayor de dispositivos tratando de conectarse a Internet. Esto conlleva al diseño de nuevos mecanismos que traten de reducir la latencia, los tiempos de espera, la pérdida de paquetes y otras características de estas conexiones. Todo ello además tratando de aumentar la seguridad en las comunicaciones, ya que cada año aumentan el número de ataques informáticos tanto a países como a empresas o a particulares.

Otra de las motivaciones para este trabajo son las limitaciones que tiene el sistema utilizado para gestionar tráfico web actualmente (HTTP/2 sobre TLS y TCP).

En el año 2013 la empresa Google presentó el protocolo QUIC [1], un protocolo de transporte que mejora estas limitaciones, además, es capaz de reducir el tiempo para establecer conexiones y mejora la latencia en las comunicaciones. El interés que Google y otras empresas han mostrado por este protocolo no ha parado de crecer desde entonces y, es por esto, que en este trabajo nos planteamos estudiarlo en profundidad.

1.2 OBJETIVO

El sistema más utilizado actualmente para gestionar el tráfico web es el uso de los protocolos HTTP/2 sobre TLS y TCP. Estos protocolos tienen varias limitaciones de prestaciones. Con el fin de solucionar estas limitaciones surge el protocolo de transporte QUIC.

Como ya se ha mencionado, QUIC fue implementado originalmente por la empresa Google, sin embargo, el IETF ha desarrollado otra versión que se encuentra en proceso de estandarización.

Objetivo: evaluar el uso del protocolo QUIC en Internet y los beneficios que aporta.

Subobjetivos para alcanzar este objetivo:

- Estudio de los distintos protocolos usados para gestionar el tráfico web durante los últimos años.

- Estudio del protocolo QUIC en detalle (tanto la versión de Google como la versión del IETF)

-Análisis comparativo de QUIC frente a los protocolos tradicionales para servir tráfico web

- Estudio experimental del protocolo QUIC: Análisis de tráfico mediante el capturador de tráfico Wireshark, uso de QUIC en distintos sitios web, compatibilidad de QUIC con navegadores, servidores web y redes de distribución de contenidos.

1.3 ESTRUCTURA DEL TRABAJO

Este documento sigue una estructura organizada por capítulos.

- En el capítulo 2, se lleva a cabo la descripción del estado del arte, en el que se muestran los protocolos más utilizados en los últimos años para gestionar el tráfico en Internet y los distintos problemas que han surgido y se han tratado de solucionar, así como el origen de QUIC. También en este capítulo se presenta el marco regulador que afecta al protocolo QUIC.
- En el capítulo 3, Diseño de QUIC, se explican las distintas versiones, y se muestran las cabeceras de la versión del IETF y la de Google, además se explica el handshake en los distintos escenarios posibles, y se describe el uso de streams y los mecanismos de control de flujo, congestión y recuperación frente a pérdidas.
- En el capítulo 4, Seguridad, se exponen los mecanismos de seguridad de QUIC y se plantean algunos de los ataques más comunes y cómo QUIC los mitiga.
- En el capítulo 5, Evaluación de QUIC, se llevan a cabo distintas evaluaciones del protocolo con la ayuda de la capturadora de tráfico “Wireshark”, se analiza el tráfico generado entre un cliente y un servidor que usan la versión de Google de QUIC y un cliente y un servidor con la versión del IETF de QUIC, también se analiza la compatibilidad del protocolo con los navegadores y páginas web más utilizados.

- En el capítulo 6, organización del proyecto, se encuentra el presupuesto del proyecto, la planificación del trabajo y el marco socio-económico de QUIC.
- En el capítulo 7, se exponen las conclusiones obtenidas acerca del diseño de QUIC y su evaluación en Internet.
- En el capítulo 8, Referencias, se encuentra la bibliografía utilizada.

Por último, en la parte final del documento hay un anexo, con un resumen del mismo, redactado en inglés.

2. ESTADO DEL ARTE

En este capítulo se analizan algunos de los protocolos utilizados para gestionar el tráfico en Internet. Se exponen las limitaciones o problemas que tienen estos protocolos y se describe cómo se pueden solucionar o mejorar. Además, se explica cómo surge QUIC y su marco regulador.

2.1 INTRODUCCIÓN A LA PILA DE PROTOCOLOS

El funcionamiento de Internet se basa en una pila de protocolos. Esta pila está formada por distintos niveles o capas, en las cuales se encuentran uno o más protocolos que dan servicio a la capa o nivel superior, y todos juntos constituyen la base de Internet hoy en día. Dicha pila de protocolos, llamada torre TCP/IP, está formada por los siguientes niveles: Aplicación, transporte, red, enlace y físico [2].

En la capa o nivel de Aplicación se definen los distintos protocolos que utilizarán las aplicaciones en sus conexiones para el intercambio de paquetes de datos. En esta capa se encuentran, por ejemplo, protocolos como FTP, HTTP, POP, DNS…

En el nivel de transporte se encuentran, entre otros, los protocolos TCP y UDP que por sus siglas en inglés significan “Transmission Control Protocol” y “User Datagram Protocol”, respectivamente. Sus principales funciones son la de multiplexar el tráfico de distintas aplicaciones sobre la torre de protocolos, y la de garantizar que las conexiones se lleven a cabo sin perder paquetes, sin errores, y que los paquetes lleguen en su orden. En este nivel se encuentra también el protocolo de seguridad TLS “Transport Layer Security”. Dicho protocolo se encarga de garantizar la seguridad en las conexiones mediante criptografía asimétrica.

En cuanto al nivel de red, se encarga de la comunicación entre los distintos routers y equipos que intervienen en las comunicaciones y establece la ruta que tomarán los paquetes de datos para llegar a su destino. Su función, por tanto, es elegir la ruta óptima entre el origen y el destino de los datos. En esta capa se encuentra el protocolo IP.

El nivel de enlace se encarga de formar los paquetes que se enviarán entre equipos vecinos. Para ello, puede basarse en sus direcciones MAC. Además, puede tener funciones de control de flujo y de detección y/o corrección de errores. Hay diversas tecnologías de nivel de enlace y físico que se usan en Internet, pero un ejemplo muy popular es Ethernet.

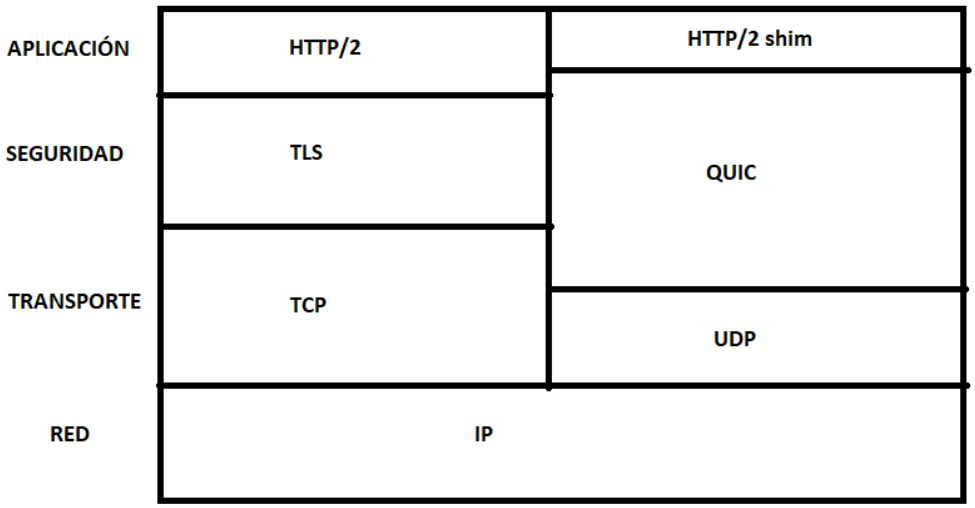


Figura 1: Pila de protocolos sin y con QUIC.

2.2 HTTP

El protocolo HTTP ("HyperText Transfer Protocol", o "Protocolo de transferencia de Hipertexto" en español) comenzó a utilizarse para la aplicación World Wide Web en 1999 [3]. Desde hace tiempo la mayoría del tráfico que circula en Internet es enviado mediante el uso de este protocolo, además, en los últimos años ha aumentado el número de aplicaciones web que lo utilizan. Por ejemplo, páginas web, streamings…

Como se ha indicado anteriormente, el protocolo HTTP se encuentra a nivel de Aplicación en la pila de protocolos. Y tiene una estructura cliente-servidor, es decir, un equipo cliente envía una petición de datos o recursos a otro equipo servidor, y este le envía una respuesta con los datos solicitados.

Un servidor es una entidad (aunque puede estar distribuida en diferentes equipos físicos), encargada de procesar las peticiones de los clientes y generar una respuesta. Sin embargo, es común que las páginas web tengan diferentes aplicaciones o contenidos que se encuentren alojados en servidores diferentes. Esto requiere de varias peticiones HTTP y,

por tanto, varias conexiones con los servidores necesarios. En la Figura 2, se muestra un ejemplo de ello.

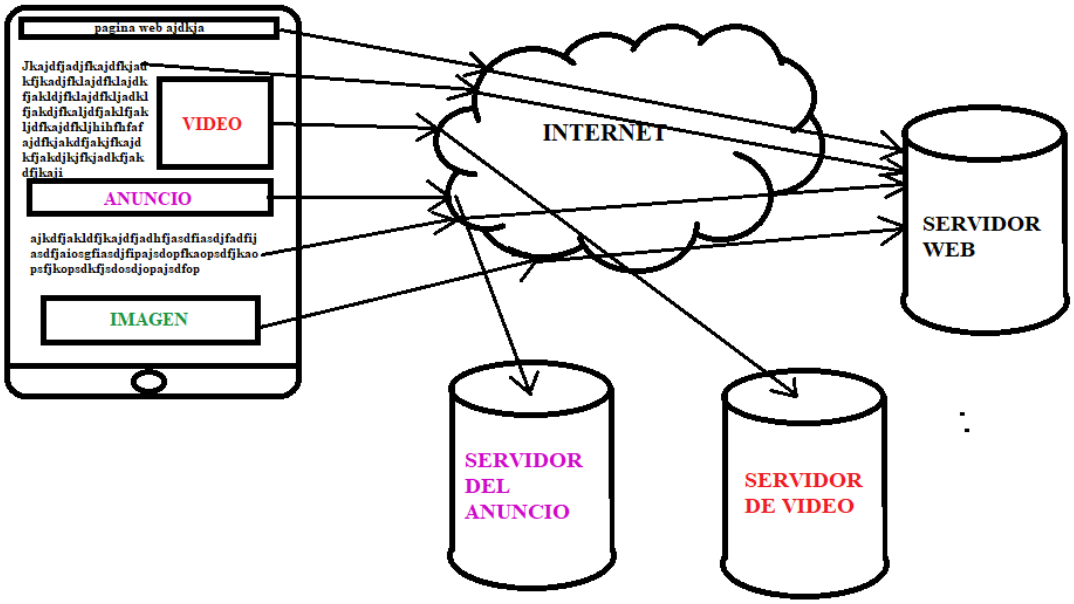


Figura 2: Página con contenido en varios servidores.

Muchas páginas web incluyen links o elementos que al activarlos redirigen al usuario a otra página. Esto se hace mediante nuevas peticiones y respuestas entre el cliente y servidor. El navegador web traduce estas peticiones y respuestas HTTP en direcciones web y muestra al cliente la nueva página solicitada.

HTTP es un protocolo sin estados, esto quiere decir que no guarda ninguna información de la conexión. Esto acarrea problemas en determinadas páginas donde el servidor requiere de datos que ha intercambiado previamente con el cliente. La solución es el uso de cookies, que son datos de pequeño tamaño que el navegador incluye en sus peticiones con un servidor que haya instalado una cookie. Esto es útil por ejemplo en una tienda online. El cliente puede navegar por la tienda y seleccionar los productos que desea comprar. Al añadir estos productos a su cesta de la compra, se añaden cookies con esa información en su navegador. De esta forma cuando el cliente accede a una nueva página para tramitar el pedido, el servidor puede acceder a sus cookies y ver que objetos quiere comprar.

2.3 HTTP y TCP

En puntos anteriores se ha dicho que HTTP es un protocolo a nivel de aplicación. Sin embargo, la multiplexación del tráfico de distintas aplicaciones sobre la torre de protocolos es imprescindible para una comunicación. Es por eso que este protocolo se sustenta sobre otro protocolo en el nivel de transporte. Los más comunes son UDP y TCP. En esta sección se va a hablar de HTTP sustentado sobre TCP.

TCP es un protocolo orientado a conexión que garantiza que los paquetes serán entregados sin errores y en su orden de envío. Para ello utiliza una serie de mecanismos, como incluir un campo checksum para detectar errores, mecanismos de asentimiento de paquetes y temporalizadores para detectar congestión o pérdidas, etc…

En las primeras versiones de HTTP (HTTP/0.9 y HTTP/1.0) se requería de un establecimiento de conexión TCP por cada pareja petición y respuesta que se enviaba. Cada creación y cierre de conexión requiere de un intercambio de mensajes entre cliente y servidor. Por tanto, crear y cerrar tantas conexiones hacía que las comunicaciones fuesen lentas [4].

Este problema se mejora en siguientes versiones de HTTP. Se explica en los siguientes puntos.

2.3.1 PAQUETES TCP

En la Figura 3 se muestra el esquema de la estructura que tiene un paquete del protocolo TCP:

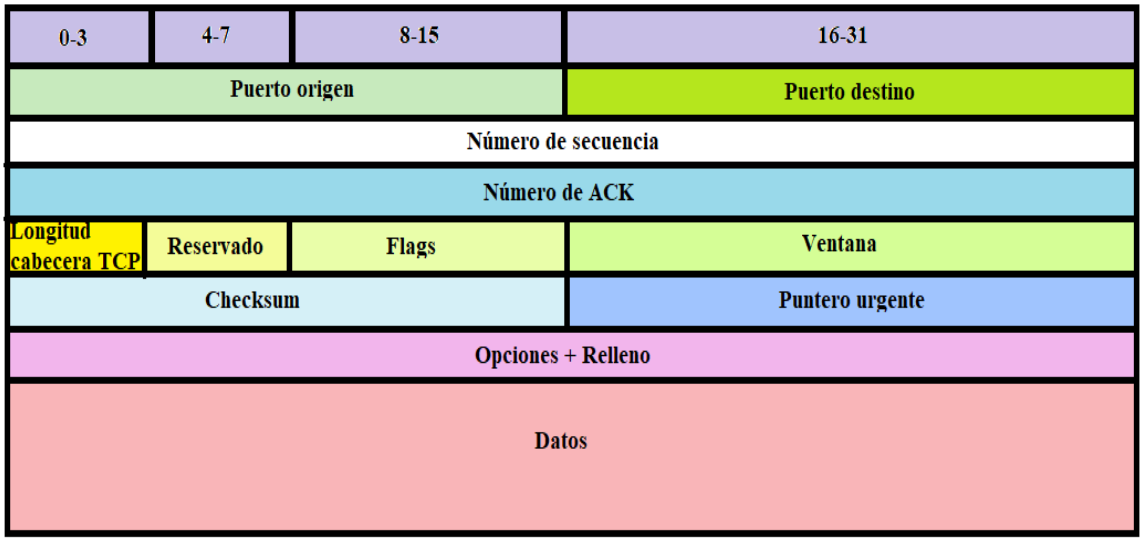


Figura 3: Paquete TCP.

Los distintos campos que se ven en la Figura 3 son [5]:

Puerto Origen: (16 bits) Corresponde con el puerto del emisor del mensaje.

Puerto Destino: (16 bits) Corresponde con el puerto del receptor del mensaje.

Número de secuencia: (32 bits) Es un número que identifica los bytes de un paquete y lo diferencia del resto. De esta forma, cuando se envíe un paquete con el número de secuencia Y que tenga X bytes, el siguiente paquete tendrá el número de secuencia Y+X. Cuando se retransmite un paquete (porque se ha perdido, por ejemplo) se hace con el mismo número que tenía el original (esto puede producir problemas como se explica más adelante cuando se habla de QUIC).

Número de ACK: (32 bits) Cuando el servidor recibe un paquete, debe confirmar al emisor que su paquete ha sido recibido con éxito. Este número sirve para identificar el asentimiento de un determinado paquete. Muestra el número del siguiente byte que el receptor espera recibir.

Longitud cabecera TCP: (4 bits) Indica el tamaño de la cabecera del paquete TCP

Reservado: (3 bits) Deben estar a 0. Este espacio está guardado para usos futuros.

Flags: (8 bits)

- CWR (“Congestion Window Reduced”): (1 bit) Se establece a 1 cuando el emisor ha recibido un mensaje TCP con el flag ECE establecido en 1, y ha activado el mecanismo de control de congestión.
- ECE (“Explicit Congestion notification-Echo”): (1 bit) Se establece a 1 cuando hay congestión en la red.
- URG: (1 bit) Se activa cuando hay prioridades, y por tanto se debe tener en cuenta el valor del campo del Puntero urgente.
- ACK: (1 bit) Indica que el campo Número de ACK es válido. Por tanto, el único mensaje que no deberá tener este campo activo es el SYN inicial.
- PSH: (1 bit) Cuando se activa este bit, indica al receptor que debe pasar todos los datos que tiene a la aplicación lo antes posible, sin esperar a recibir el resto de datos que están en camino.
- RST: (1 bit) Cuando se activa este bit se cierra la conexión de forma abrupta. Esto se produce cuando hay determinados errores, por ejemplo, que se reciba una solicitud de conexión cuando no hay ninguna aplicación esperando por ella.
- SYN: (1 bit) Se activa para indicar que se trata del primer mensaje SYN, de cada lado, que inicia la conexión.
- FIN: (1 bit) Cuando se establece a 1, indica al receptor que el emisor quiere finalizar la conexión y se puede liberar.

Ventana: (16 bits) Indica el número máximo de bytes que pueden ser recibidos, ya que si se envían más datos no caben en el buffer del receptor y tendría que descartar paquetes. Este mecanismo de control de flujo se explica más adelante.

Checksum: (16 bits) En éste cálculo se guarda una suma aritmética de los bytes del mensaje. Sirve para detectar posibles errores.

Puntero urgente: (16 bits) Indica el tamaño de los bytes que son urgentes. Para ello muestra el número de bytes que hay desde el número de secuencia hasta el último urgente o hasta el primer byte no urgente.

Opciones + Relleno: (variable) El campo de Opciones sirve para añadir opciones de cabecera adicionales que no están incluidas en las explicadas anteriormente. Posteriormente se añade un campo de Relleno con tantos bits a 0 como sean necesarios para que la cabecera tenga un tamaño múltiplo de 32 bits.

Datos: Es la información que el cliente envía al servidor.

2.3.2 THREE WAY HANDSHAKE EN TCP

El establecimiento de conexión entre un cliente y un servidor requiere de un intercambio de mensajes entre ambos. Este intercambio de mensajes se denomina handshake. Hay varios tipos de establecimiento de conexión, TCP utiliza el llamado saludo de 3 vías (Three Way Handshake en inglés) [5]. En la Figura 4 se muestra cómo es este intercambio:

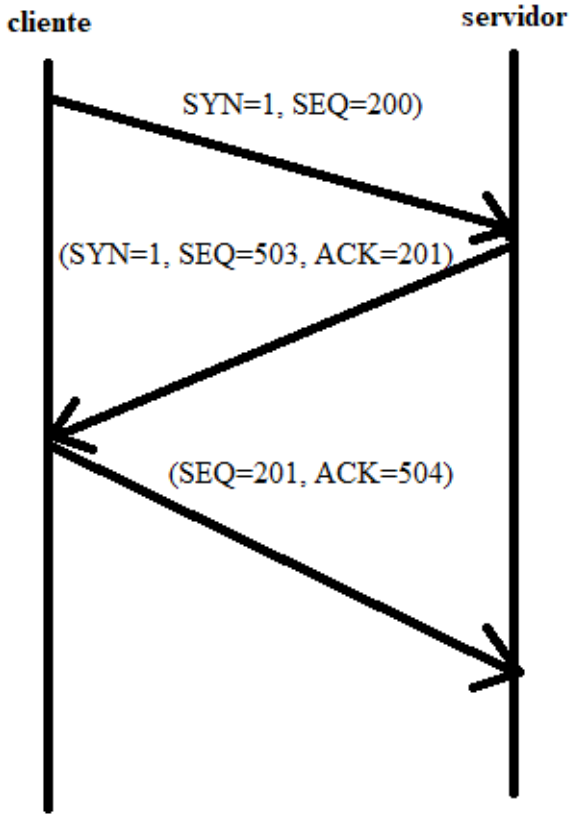


Figura 4: Three Way Handshake en TCP.

Como se puede ver, el establecimiento de conexión lo inicia el cliente y requiere de 3 mensajes:

1) El cliente envía un mensaje de sincronización (SYN), en dicho mensaje se encuentra el número de secuencia (elegido por el cliente, por ejemplo 200).
2) El servidor procesa la solicitud de inicio de conexión y, si acepta establecerla: responde al cliente con otro mensaje SYN, en el cual indica su número de

secuencia (503, no tiene por qué corresponder con el número de secuencia del cliente) y en su campo ACK el número de secuencia del mensaje que ha recibido +1 (indicando así que ha recibido correctamente el mensaje de SYN y está esperando a recibir el byte 201).

3) Al recibir este mensaje el cliente ya sabe que la conexión está abierta, y envía otro paquete para confirmar al servidor que la conexión se ha establecido. Este paquete tiene el número de secuencia anterior+1 (201, que es el que le pedía el servidor) y un ACK con el número de secuencia del servidor +1 (504, indicando que recibió el 503 y el siguiente byte que espera recibir es el 504). Este paquete puede llevar ya datos del cliente, pero incluso si no contiene datos, hay que enviarlo porque la recepción de este mensaje permite al servidor confirmar que la conexión está abierta.

Este sistema de inicio de conexión requiere de 1 RTT, antes de poder enviar datos, que es el tiempo que transcurre entre que el cliente envía su primer mensaje SYN y recibe el primer mensaje SYN del servidor. (RTT es un acrónimo de “Round Trip Time”, consiste en el tiempo que transcurre desde que se envía un paquete, hasta que se recibe el paquete de respuesta del otro lado).

2.3.3 CONEXIÓN E INTERCAMBIO DE MESAJES CON HTTP Y TCP

Para realizar una petición HTTP, primero es necesario establecer una conexión TCP. Para ello, se utiliza el método de Three way handshake que se explica en el punto anterior. Una vez establecida la conexión TCP, el cliente puede utilizarla para realizar las peticiones HTTP.

Las peticiones HTTP tienen la siguiente estructura (en la Figura 5 se muestran la estructura arriba, y un ejemplo de uso debajo):

{Método} / {Versión}
{Cabeceras}

GET / HTTP/1.1
Host: google.com

Figura 5: Estructura y ejemplo de una petición HTTP.

El campo {Método} corresponde con la operación que el cliente desea realizar. Por ejemplo, un método {GET}, sirve para solicitar recursos.

En {cabeceras} se incluyen campos opcionales para completar la petición, por ejemplo, el puerto al que realiza la petición, la dirección URL del recurso que se solicita, el idioma…

Se pueden encontrar todas las posibles cabeceras y su funcionamiento en el documento [4].

El servidor contesta a estas peticiones con mensajes denominados respuestas. La estructura de las respuestas es muy similar:

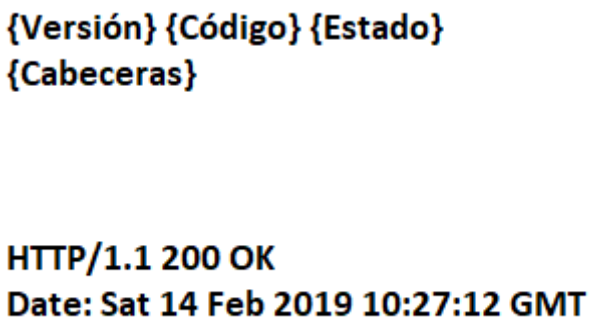


Figura 6: Estructura y ejemplo de una respuesta HTTP.

El campo {Código} indica si la petición ha tenido éxito o no y por qué motivo. El campo {Estado} es un mensaje con una breve descripción del código, tal y como se muestra en el ejemplo de la Figura 6.

En resumen, primero se establece la conexión TCP. Después, sobre esta conexión, se produce el intercambio de peticiones y respuestas HTTP necesarias para acceder al recurso.

2.3.4 PROBLEMAS DE ESTE SISTEMA

Hasta ahora se ha explicado el proceso necesario para establecer una conexión que permita acceder a un recurso utilizando HTTP junto con TCP. Se ha mostrado que primero es necesario establecer una conexión TCP, lo cual tarda 1 RTT. Después se envía

una petición HTTP y se espera a una respuesta por parte del servidor, lo cual conlleva más retardo (1 o más RTTs).

Por tanto, si queremos acceder a un recurso como el que se muestra en Figura 2, en el cual, la información que queremos se encuentra en 3 servidores diferentes, necesitamos establecer 3 conexiones TCP distintas y 3 intercambios de peticiones y respuestas HTTP. Esto implicaría 6 RTT sólo en establecer conexiones, antes de ser capaces de enviar datos.

Este problema mejora utilizando conexiones TCP persistentes. Se explicará en el punto 2.5.1.

2.4 TLS

En los últimos años el número de ataques informáticos ha aumentado considerablemente, y se estima que siga aumentando.

En la Figura 7 se puede ver como entre enero de 2016 y octubre de 2017 el volumen de malwares detectados aumentó en 11 veces su volumen.

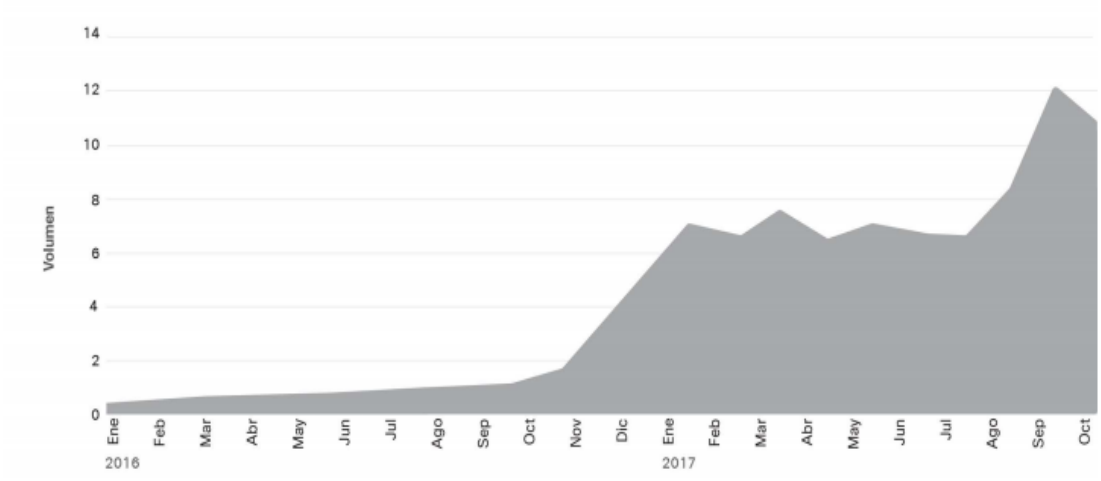


Figura 7: Volumen de malware detectado entre enero 2016 y octubre 2017. Tomada de [6].

Además, estos años ha aumentado también el uso de aplicaciones bancarias, tarjetas de crédito online y otros tipos de datos de alto valor. Todo ello ha hecho necesario aumentar la seguridad en Internet.

TLS es un protocolo de seguridad utilizado junto con TCP y HTTP para garantizar la seguridad en las comunicaciones. Ofrece servicios de autenticidad, integridad,

encriptación y uso de claves criptográficas. A partir de ahora, el uso de estos tres protocolos será referido en este documento como HTTPS.

Los distintos métodos de cifrado se pueden clasificar en dos grandes grupos: criptografía simétrica y asimétrica. En la criptografía simétrica hay una única clave que sirve tanto para encriptar como para desencriptar. Sin embargo, en la criptografía asimétrica se diferencia entre una clave pública y una privada. Es decir, en asimétrica el mensaje se cifra y se descifra con claves diferentes. Esto implica que el cifrado mediante criptografía asimétrica es más seguro, pero requiere de más tiempo de procesamiento.

2.4.1 TLS HANDSHAKE

Para realizar una petición a un servidor en HTTPS, primero es necesario establecer la conexión TCP como se vio anteriormente. Después se realiza la conexión TLS y por último la conexión HTTP.

Durante el establecimiento de conexión TLS se llevan a cabo las negociaciones criptográficas que usaran ambas partes. (Diffie Hellman, RSA, Curvas Elípticas…). Esta negociación se realiza mediante criptografía asimétrica, ya que el cliente y el servidor no han establecido una comunicación previa, y no disponen de un canal seguro por el cual enviarse una clave simétrica. En esta fase, cliente y servidor establecen un canal seguro y acuerdan una única clave de criptografía simétrica. De esta forma, el resto de la comunicación será encriptado únicamente con criptografía simétrica que es más rápida [7].

1) En primer lugar, el cliente envía un mensaje “Client HELLO” para indicar al servidor que quiere establecer una conexión TLS.
2) El servidor le responde con otro mensaje “Server HELLO” indicándole que está dispuesto a establecer la conexión, y a continuación le envía su certificado digital y un asentimiento. El certificado digital contiene la clave pública del servidor.
3) El cliente genera una clave simétrica y la cifra con la clave pública que ha obtenido del servidor y lo envía en el mensaje “Key Exchange”. Seguido envía un mensaje (“Change Cipher Spec”) indicando que ya pueden cambiar a criptografía simétrica. Por último, envía un mensaje (“Client Finished”) indicando que ya ha acabado su parte de la autenticación.

4) El servidor desencripta el mensaje Key Exchange que ha recibido utilizando su clave privada de criptografía asimétrica. Y en este punto ambas partes conocen la única clave que se usará a partir de ahora para el intercambio de datos utilizando criptografía simétrica. El servidor envía un mensaje (“Change Cipher Spec”) para indicar que a partir de ahora va a usar criptografía simétrica. Y un mensaje (“Server Finished”) indicando que ya ha terminado su parte de la negociación.

Estos intercambios se pueden ver en la Figura 8:

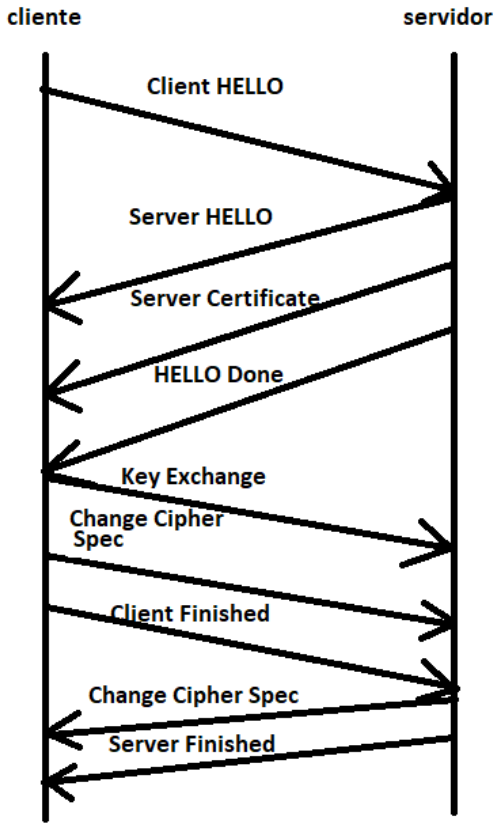


Figura 8: Handshake TLS 2RTT.

En conclusión, estos intercambios para negociar una clave de criptografía simétrica requieren de 2 RTT. Si sumamos estos 2 RTT a los 2 RTT que tenía el protocolo HTTP junto con TCP, vemos que el protocolo HTTPS tarda 4 RTT antes de poder enviar datos al otro extremo.

2.5 PROBLEMAS ADICIONALES

Como se ha visto hasta ahora, hay varias limitaciones en este sistema. Seguridad, demasiados RTT… Además de esto hay otros problemas, como por ejemplo el bloqueo de una descarga por un contenido concreto. Esto consiste en lo siguiente:

En HTTP/1.1 y versiones anteriores, no se permite bajar contenidos en paralelo sobre una misma conexión TCP. Por tanto, hay dos opciones, bajarlos usando varias conexiones TCP en paralelo o en serie, o bajarlos en serie sobre una única conexión TCP (TCP persistente, que se explica más adelante). Tener varias conexiones TCP abiertas en paralelo es costoso, y en general, los servidores limitan el número de conexiones TCP en paralelo con un mismo cliente. El envío de contenidos en serie, en una conexión TCP o en varias, genera bloqueos. Si hay un problema en uno de esos contenidos y se bloquea su descarga, se bloquearán también todas las descargas que vienen a continuación. A continuación, se verá como este problema se soluciona con HTTP/2, que permite descargas en paralelo sobre una misma conexión TCP. Sin embargo, esta solución es parcial, ya que, aunque HTTP/2 permita la descarga en paralelo, TCP no lo permite, está obligado a entregar los bytes en orden. Por tanto, si se pierde o se bloquea un paquete con un determinado contenido, afecta a todos los demás paquetes con contenido común, ya que no pueden ser entregados hasta que se recupere el paquete perdido. Se verá cómo esto se soluciona con QUIC. Este tipo de problemas (denominados en inglés “Head Of Line Blocking”) se explicarán más adelante.

2.6 SOLUCIONES

Se han desarrollado nuevas versiones de HTTP, así como otros mecanismos que tratan de solucionar los problemas que se han destacado en los apartados anteriores.

2.6.1 TCP PERSISTENTE

Hasta ahora se ha visto que si se quiere acceder a una página web con un elemento HTML y 3 objetos diferentes (vídeos, anuncios…) es necesario establecer una conexión TCP diferente para cada objeto. Este problema se soluciona con el uso de TCP persistente. Este protocolo permite dejar la sesión TCP abierta y realizar las diferentes peticiones y respuestas a través de una única sesión TCP. Esto tiene sus ventajas y sus inconvenientes.

Las ventajas son; para bajar varios contenidos, se ahorran todos los tiempos de establecimiento de conexión para cada contenido (lo que es muy significativo porque típicamente hay muchos contenidos en una página web). También, por cómo funciona el control de congestión de TCP, mejora el throughput obtenido en la bajada de contenidos.

Sin embargo, TCP persistente tiene algunos inconvenientes. Los servidores tienen unos recursos limitados, y debido a que las conexiones TCP son costosas, necesitan limitar el número de conexiones TCP que mantienen abiertas al mismo tiempo para no saturarse. Por tanto, mantener las conexiones abiertas puede limitar el número de usuarios que acceden al recurso. Este problema siempre se puede afrontar cerrando desde el servidor conexiones TCP abiertas y no en uso si lo necesita por falta de recursos.

En general, los beneficios hacen que se prefiera el uso de TCP persistente al no persistente.

Se puede consultar más información acerca de TCP persistente aquí [8].

2.6.2 TLS 1.3

TLS 1.3 es la cuarta versión del protocolo TLS e incluye una serie de mejoras frente al protocolo inicial explicado anteriormente. Aparte de corregir varios aspectos de seguridad que la versión anterior no contemplaba, reduce el número de intercambio de mensajes necesarios en el handshake.

El nuevo handshake para establecer los parámetros de seguridad en TLS 1.3 tiene la siguiente forma [9]:

1) En el primer paso, el cliente envía un mensaje “Client HELLO” seguido de un mensaje “Cypher Suite” que consiste en una lista con los distintos algoritmos de encriptado y funciones Hash que soporta. Además, envía un mensaje “Key Share” que servirá al servidor para conocer la misma clave simétrica, que ambos usarán.
2) El servidor contesta con un mensaje “Server HELLO”, que contiene el algoritmo elegido de la lista que ha recibido, seguido de un mensaje “Key Share” que necesitará el cliente para generar la clave simétrica, un mensaje “Server Certificate” con su certificado digital, y un mensaje “Server Finished” indicando que por su parte ya ha terminado el handshake.

3) El cliente al recibir esto, comprobará el certificado el servidor, generará la misma clave simétrica que el servidor y contesta enviando un mensaje “Client Finished” indicando que por su parte ha terminado el handshake. En este punto ya se pueden enviar datos de forma segura.

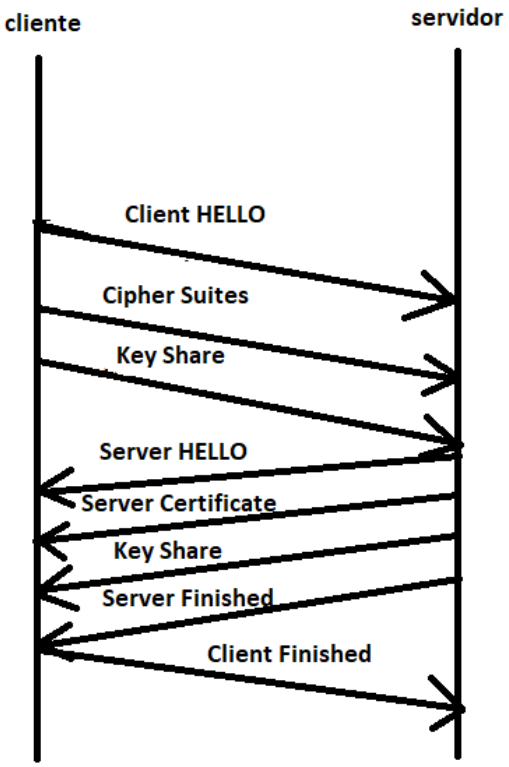


Figura 9: Handshake TLS 1.3.

En conclusión, con la nueva versión de TLS 1.3 sólo se requiere de un RTT para negociar los parámetros de seguridad, en lugar de 2 como antes. Por tanto, se pueden empezar a enviar datos después de 2 RTT (uno de TCP, y otro de TLS 1.3).

2.6.2.1 TLS 1.3 CON RESTABLECIMIENTO DE SESIÓN

Cuando se ha establecido una conexión previa entre un cliente y un servidor de la forma mostrada anteriormente, ambos pueden guardar esa clave simétrica. De esta forma si después de una conexión el cliente quiere volver a enviar datos al servidor, podrá restablecer su sesión anterior, autenticando al servidor y encriptando sus mensajes con la clave que habían acordado previamente. Esto implicaría que se puede intercambiar datos

de forma segura entre cliente y servidor con 0RTT adicionales (siempre y cuando se haya establecido una conexión previa que habría tardado 1RTT). De modo que restablecer una conexión para enviar contenidos HTTP usantdo TLS 1.3 sólo supone 2 RTT (frente a los 4 que suponían las primeras versiones de este protocolo).

El restablecimiento de conexión, por tanto, es más rápido que el primer inicio de conexión, pero a la vez es menos seguro, ya que, si la clave se ha comprometido, un posible atacante podría utilizarla para enviar nuevos datos al servidor.

Otro problema es que esta reanudación de sesión 0RTT es vulnerable frente a ataques por repetición. Estos ataques consisten en que, si un posible atacante es capaz de escuchar la conexión cifrada entre cliente y servidor, y además conoce los datos que envió el cliente en la primera solicitud, puede copiar estos datos, enviárselos al servidor y establecer una nueva conexión suplantando su identidad. El servidor sabrá que esta solicitud es repetida, pero como no tiene forma de ver el origen real del cliente, el atacante podrá suplantarle y llevar a cabo el ataque con éxito.

Actualmente hay varios mecanismos que mitigan el problema de los ataques por repetición [9].

2.6.3 HTTP/2

Otra solución al problema de establecer una conexión por cada recurso se incorpora en la versión HTTP/2. En la Figura 2 se muestra cómo una página puede estar compuesta por varios objetos, sin embargo, es un ejemplo muy básico. Actualmente las páginas web pueden tener decenas de objetos (imágenes, vídeos, JavaScript, código HTML…). En las versiones anteriores de HTTP que se han analizado se enviaba una petición a cada objeto y se esperaba a la respuesta antes de enviar la siguiente solicitud. Con la llegada de TCP persistente, se pueden enviar contenidos en serie uno detrás de otro por una misma conexión TCP. Pero esto tiene dos inconvenientes: el primero, es que, al mostrar la página web en el navegador, los contenidos se van mostrando uno detrás de otro y no todos a la vez, lo que puede ser menos agradable para el usuario. El segundo, es el bloqueo “Head of Line Blocking” explicado antes.

La solución que aporta HTTP/2 viene dada por un nuevo método de multiplexación [10]. La multiplexación permite combinar varias señales en un mismo canal. Por tanto, en una misma conexión se pueden llevar a cabo varias solicitudes HTTP en paralelo, en lugar de

hacerlas en serie como pasaba en HTTP/1.1, esto es similar al problema que se comentó con TCP y TCP persistente. Además, soluciona parcialmente el problema de bloqueo "Head of Line Blocking", ya que sólo se soluciona si se produce a nivel de aplicación, pero no a nivel de red/transporte.

2.7 QUIC

En los puntos anteriores se ha explicado el funcionamiento de los protocolos TCP, TLS y HTTP algunos de los problemas que plantean y diferentes soluciones. Sin embargo, siguen consumiendo muchos recursos de CPU, generando demasiados establecimientos de conexión y un elevado número de RTTs por cada conexión. Este problema tiene más importancia en los últimos años ya que están aumentando el número de comunicaciones que requieren de una baja latencia, como por ejemplo los streamings.

Con la idea de seguir con las mejoras vistas en los protocolos anteriores surge QUIC. QUIC es un protocolo desarrollado por la empresa Google, que reemplaza algunas de las capas de la pila de protocolos. QUIC se sustenta sobre UDP en lugar de TCP y no requiere de otro protocolo de seguridad como TLS, ya que el propio protocolo contiene diferentes medidas para ofrecer el mismo nivel de seguridad (de hecho, se verá posteriormente que tiene un mayor grado de seguridad).

QUIC surge con la finalidad de reducir la latencia en las comunicaciones HTTP a través de Internet. Disminuye el número de paquetes que tienen que intercambiarse dos equipos para iniciar una conexión, y por tanto, permite que las conexiones se establezcan más rápido. Para reducir este número de paquetes que se intercambian en el handshake, se basa en credenciales que tiene almacenadas de una conexión previa entre dichos equipos y elimina los paquetes del handshake que sean redundantes o innecesarios. Esto se explicará más detalladamente en el punto 3.7.

Además de esto, cada año aumenta el número de ciberataques que se realizan y la seguridad informática cada vez cobra más valor. El encriptar paquetes y llevar a cabo conexiones seguras causa retardos. El sistema actual de TLS y TCP es válido y aporta un alto nivel de seguridad. Sin embargo, tiene limitaciones, ya que, como se ha descrito, necesita varios intercambios de paquetes en los handshake (TCP y TLS) para establecer una comunicación y garantizar dicha seguridad. Esto ha conducido a la necesidad de nuevos protocolos como el que se está explicando, que permiten reducir el número previo

de intercambios de paquetes y mejorar la sensación de usuario sin renunciar a la seguridad de la conexión.

2.8 INVESTIGACIÓN SOBRE QUIC

Debido a las ventajas que QUIC proporciona, muchas empresas y particulares están siguiendo muy de cerca la evolución de este protocolo. Ya hay varios trabajos investigando el uso de QUIC en internet, muchos desde el punto de vista del tráfico en routers. Un trabajo interesante sobre el uso de QUIC en la red es [11]. Nosotros queremos contribuir al esfuerzo de entender el uso actual de QUIC desde un punto de vista más práctico con las herramientas disponibles.

2.9IMPLEMENTACIÓN DE QUIC E IMPACTO EN SU DESPLIEGUE

Los sistemas operativos actuales tienen varias capas con el fin de evitar que archivos maliciosos dañen el hardware de nuestro equipo. De esta forma se puede diferenciar entre espacio de núcleo o kernel, el cual se encarga de gestionar los recursos de Hardware de nuestro equipo y proporciona una interfaz sencilla para el usuario; y el espacio de usuario, en el cual se encuentran la mayoría de las aplicaciones o programas que utiliza el usuario. La diferencia es que los programas o aplicaciones que se encuentran en el espacio de usuario no pueden interactuar de forma directa con el hardware, y obtienen acceso al hardware pidiendo servicios al núcleo (API de llamadas del sistema). Por otro lado, los situados en el espacio de núcleo tienen acceso directo.

En el punto anterior se ha visto la torre de protocolos. Esta torre suele implementarse en el espacio de núcleo hasta el nivel de transporte. De este modo, se puede acceder a ella desde cualquier aplicación usando las librerías del sistema operativo. El inconveniente es que para cualquier modificación o actualización que se desee hacer sobre ellas, primero deberá estandarizarse y luego ser implementadas por los distintos desarrolladores de sistemas operativos. Por último, podrán llegar al usuario final actualizando su sistema operativo.

La empresa Google abarca un sector muy amplio, y está presente tanto en el ámbito de gestores de contenido como en el de navegadores web (y otros clientes incluyendo aplicaciones para móviles). En vista de las limitaciones de los sistemas actuales decidió desarrollar un nuevo protocolo de transporte. Al ser una empresa tan amplia, controla

tanto los servidores donde se almacena contenido (Gmail, YouTube…), como los clientes (Google Chrome, buscador de Google de dispositivos Android, YouTube App en dispositivos móviles…).

Google es la empresa desarrolladora y propietaria del navegador "Google Chrome". En marzo de 2019 se calculó que el porcentaje de usuarios de Internet que utilizaban Chrome, como su navegador web, era de 67,88%. El segundo navegador más utilizado fue "Mozilla Firefox" con un 9,27% [12]. Por tanto, Google domina el mercado de navegadores web y por tanto tiene control sobre el cliente y esto facilita la implementación de QUIC entre estos clientes y sus servidores.

Es gracias a esto que Google ha podido desarrollar su protocolo implementado a nivel de aplicación y utilizando UDP. Esto es muy importante, ya que, si no controlase ambos extremos, tendría que desarrollar este protocolo en nivel de transporte, estandarizarlo y que los desarrolladores de sistemas operativos quieran implementarlos en sus sistemas. Otra opción sería hacerlo también a nivel de aplicación y que el resto de empresas en el mercado decidan añadirlo a su navegador. Realmente la estrategia de Google ha funcionado por su posición en Internet. Esto es tan así, que todavía hoy, el tráfico QUIC en Internet casi en su totalidad involucra a servidores y clientes de Google (como veremos en el capítulo 5), y será el proceso de estandarización el IETF lo que posibilite que se extienda más allá de Google. Pero sólo con Google, el impacto en el tráfico es muy importante.

En conclusión, Google desplegó QUIC en el espacio de usuario, esto facilita su uso para varias aplicaciones y lo hace más flexible, ya que permite añadir modificaciones o actualizaciones más fácilmente. Esto tiene también sus inconvenientes. Si Google genera muchas implementaciones o actualizaciones de QUIC se pueden producir incompatibilidades.

QUIC es un protocolo con código abierto, es decir, el código fuente en C++ que utilizan las aplicaciones que se mencionan anteriormente están publicados y se puede usar y modificar libremente.

2.10 MARCO REGULADOR

Este trabajo incluye una recolección de información teórica y hace un trabajo experimental. En cuanto a la recolección de información, se utilizan fuentes públicas.

Parte de esa información proviene de documentos y estándares que son abiertos. En cuanto a la parte experimental, no maneja ninguna información privada. Las peticiones que se llevan a cabo con servidores web para determinar si son compatibles o no con el protocolo QUIC se hacen de una en una y manualmente, como un usuario cualquiera, por lo que no se causa ningún tipo de sobrecarga sobre los servidores. Además, el tráfico que se captura en las comunicaciones siempre es propio, no de terceros usuarios.

Más adelante se explica la diferencia entre las dos versiones más importantes de QUIC, la desarrollada por Google y la desarrollada por el IETF. Sin embargo, el marco regulador es muy distinto para ambas versiones.

Si cada fabricante creara sus propios protocolos para conectarse a la red, podrían producirse problemas de incompatibilidad y no sería posible comunicarse entre dispositivos de distintos fabricantes. Es por este motivo, que es necesaria la presencia de una organización que lleve a cabo unos estándares o reglas, para que todos los dispositivos utilicen los mismos protocolos. De esta forma, no hay conflictos entre equipos de diferentes fabricantes al conectarse a la red. La organización más grande en el mundo dedicada a la realización de estándares de protocolos para usar en Internet es el IETF.

Por tanto, la versión de QUIC desarrollada por Google, no se ve afectada por esta regulación, ya que, al controlar un buen número de clientes y servidores, ha desplegado este protocolo a nivel de aplicación y por tanto no necesita de otras empresas u organismos para su estandarización, siempre y cuando la conexión se lleve a cabo entre un cliente de Google y un servidor de la misma compañía. Lo único que se necesita es un método para saber cuándo se puede usar QUIC, o si la comunicación se produce con un cliente o servidor que no es compatible (con lo que habría que comunicarse usando HTTP/TLS/TCP).

Esta situación permite a Google ser muy flexible en la evolución del protocolo según sus intereses, pero no favorece la adopción de QUIC por terceros. Es por ello que el interés en el protocolo ha llevado, con el apoyo de Google, al proceso de estandarizarlo en el IETF, El proceso de estandarización del IETF es abierto, lo que quiere decir que todos los interesados pueden contribuir para que el protocolo cumpla con sus necesidades, y la adopción de los estándares se produce por consenso.

Un estándar proporciona garantías de compatibilidad y de protección de los derechos de la mayoría, que no se tienen cuando se trata de un protocolo desarrollado individualmente

y manejado por una empresa. Es por ello que un estándar favorece la adopción por distintos fabricantes que lo pueden desplegar en sus equipos.

El Working Group de QUIC del IETF es [13]. Ahí, en el apartado de documentos, se encuentra también una lista con los drafts que ha adoptado el WG y se encuentran en proceso de estandarización, y los drafts que contienen propuestas individuales acerca de QUIC. Una vez que un WG ha adoptado un draft y ha sido revisado, se envía a un editor de RFC, el cual se encargará de revisar que se encuentra en un formato coherente y, tras realizar algunos cambios si son necesarios y consultarlos con los autores, se procede a la publicación del documento como RFC.

Algunos drafts importantes del WG de QUIC son [14], en el que se detalla un mapeo de la semántica de HTTP sobre QUIC, y [15], en el que se explica la manejabilidad del protocolo QUIC.

2.11 CONCLUSIONES DEL CAPÍTULO

A medida que ha avanzado la tecnología, ha sido necesario reemplazar o actualizar los protocolos que gestionan el tráfico de Internet, tanto para solucionar problemas o limitaciones, como para llevar a cabo comunicaciones más rápidas y con menos latencia. Esto ha motivado el desarrollo del protocolo QUIC, que mejora las comunicaciones. El uso de este protocolo es posible gracias a que Google ha decidido implementar su protocolo a nivel de aplicación, ya que controla tanto clientes como servidores.

3. DISEÑO DE QUIC

En este capítulo se explica el diseño técnico del protocolo QUIC. Para ello se describen las diferentes versiones y se explican sus cabeceras. Además, se muestran los distintos handshake que puede realizar el protocolo, según el escenario, y se muestra una comparación entre el tiempo requerido para establecer conexiones con este sistema, y algunos de los sistemas explicados en el capítulo anterior.

También se describe el uso de streams en QUIC y se describen algunos de los mecanismos que tiene QUIC para controlar el flujo y la congestión de sus comunicaciones.

3.1 QUIC DE GOOGLE Y QUIC DEL IETF

Como se ha expuesto en el primer capítulo, QUIC es un protocolo desarrollado por Google. Sin embargo, atraídos por sus ventajas frente a TCP y HTTP/2, el IETF está trabajando en su propia versión para estandarizar el protocolo. Sin embargo, actualmente, sólo se encuentra en uso el desarrollado por Google debido a lo mencionado en el punto 2.8.

Tanto el protocolo desarrollado por Google, como el desarrollado por el IETF son muy parecidos y conllevan los mismos resultados en cuanto a latencia, velocidad, seguridad…

En el IETF se está hablando de añadir nuevas funcionalidades, esto es bastante habitual en procesos de estandarización, ya que al estar formados por personas que pueden pertenecer a distintas empresas, hay distintas partes tratando de añadir funcionalidades que les interesen. Al no estar aún estandarizado, hay muchas funcionalidades que no están concretadas aún, por tanto, este documento se centra en la funcionalidad básica.

A su vez, dentro del protocolo de IETF, existen varias versiones, ya que a media que avanza el protocolo, surgen nuevas ideas, requisitos... Lo mismo ocurre con el protocolo de Google.

Para determinar qué versión se utilizará en la comunicación se lleva a cabo una negociación.

3.2 NEGOCIACIÓN DE VERSIÓN

Tanto el protocolo de Google como el de IETF tienen en su cabecera un campo llamado "Versión" de 32 bit. En este campo se indica la versión del protocolo que se está utilizando. En el protocolo del IETF este campo se encuentra únicamente en la cabecera larga. Se explicará más detalladamente en el siguiente capítulo.

Cuando uno de los dos extremos recibe un paquete que no entiende, porque no soporta esa versión, responde al otro extremo con un paquete de Negociación de versión. Dicho paquete consiste en un paquete en el cual el campo "Versión" de su cabecera, tiene el valor 0x00000000 (32 bit a cero). Este valor está reservado y sólo se utilizará para indicar que es necesario negociar la versión. Además de esto también incluye una lista de las distintas versiones que soporta [16].

A continuación, se muestra una imagen de un paquete utilizado en la versión de QUIC del IETF para llevar a cabo una negociación de versión:

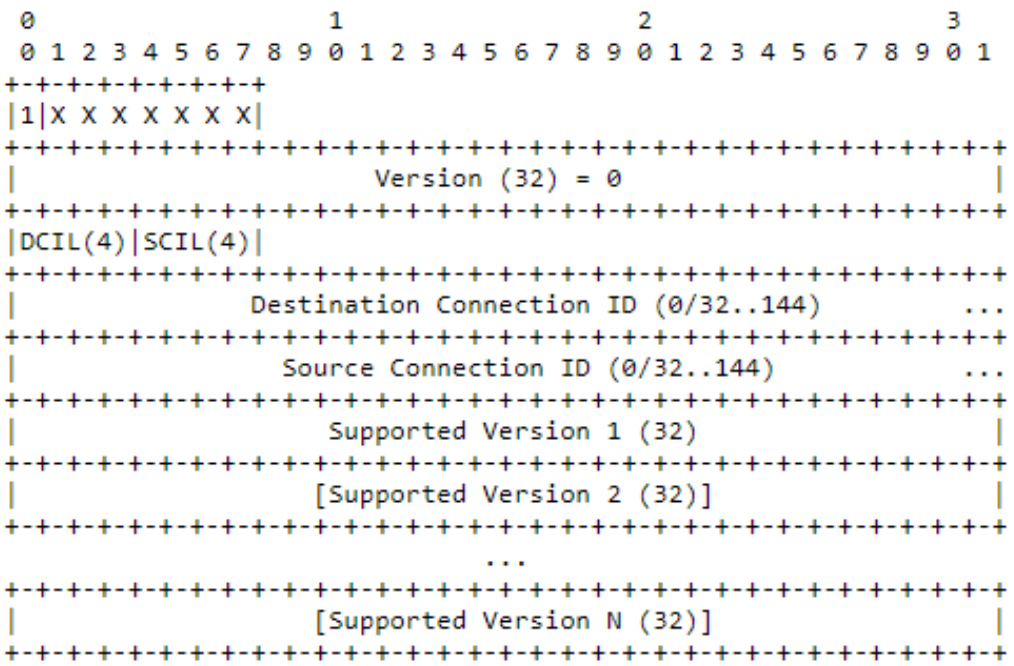


Figura 10: Paquete de negociación de versión IETF. Tomada de [16].

Los campos “Destination Connection ID”, “Source Connection ID”, “DCIL” y “SCIL” son comunes con la cabecera larga estándar en los paquetes QUIC del IETF, por tanto, se explicarán en el siguiente capítulo. Los campos que son propios de esta cabecera son [16]:

Primeros 8 flags (8 bit):

- Primer bit: activado siempre, para indicar que se trata de una cabecera larga.
- Del bit 2 al 8: no tienen uso aún.

Version: (32 bit) Como se explicó anteriormente, contiene el valor 0x00000000 para indicar que se trata de un paquete de negociación de versión.

Supported Version 1: (32 bit) Contiene el valor identificador de la primera versión que soporta.

Supported Version 2: (32 bit) Contiene el valor identificador de la segunda versión que soporta.

Y así sucesivamente, hasta llegar a la última versión que soporta (N).

Este tipo de paquetes no utilizan protección de identidad o confidencialidad. Sin embargo, el extremo receptor del paquete es el que se encargará de autenticarlo como parte de su proceso de establecimiento de conexión.

Un extremo debe rechazar paquetes que tengan una lista de versiones soportadas truncada, ya que un posible atacante podría eliminar las últimas versiones de su lista para llevar a cabo un ataque por degradación de versión (explicado más detalladamente en el capítulo 4, dedicado a la seguridad).

3.3 ESTRUCTURA PAQUETES QUIC (IETF)

Como se indica en el punto anterior, el IETF está estandarizando su propia versión del protocolo QUIC. En esta sección se analizará la estructura de la cabecera de los paquetes según la versión del IETF.

Existen dos estructuras diferentes para las cabeceras de los paquetes QUIC. Una cabecera larga y una cabecera corta. La cabecera larga se utiliza en los primeros paquetes, que sirven para iniciar la conexión, mientras que el resto de los paquetes, que sirven para intercambiar información, usan la cabecera corta [17].

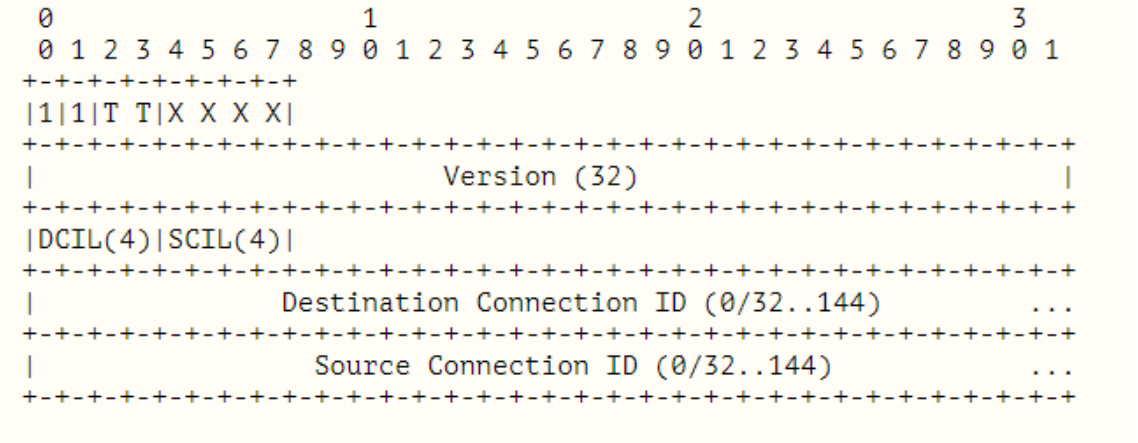


Figura 11: Cabecera larga paquete QUIC. Tomada de [17].

Como se puede ver en la Figura 11, la cabecera larga está formada por los siguientes campos:

Primeros Flags: (8 bits)

- Primer bit: Cuando tiene un 1, indica que se trata de un paquete con cabecera larga.
- Segundo bit: Siempre está establecido a 1 en esta versión.
- Tercer y cuarto bits (T): Contienen un valor que representa el tipo de paquete largo del que se trata. El tipo de paquete puede ser:

  00 -> Inicial (se utiliza para los primeros intercambios de paquetes cifrados para negociar la clave de sesión).

  01 -> 0-RTT (se utiliza para el primer paquete que restablece una conexión que finalizó recientemente).

  02 -> Handshake (se utiliza para llevar a cabo el handshake entre cliente y servidor).

  03 -> Retry (Se utiliza cuando un servidor quiere reintentar una conexión fallida).
- Del quinto al octavo bit: Son específicos para cada tipo de paquete de la lista anterior.

Version: (32 bit) En este campo se encuentra el valor que representa la versión del protocolo que se está utilizando en la comunicación.

DCIL: (4 bit) Estos 4 bit indican la longitud del campo “Destination Connection ID”.

SCIL: (4 bit) Estos 4 bit indican la longitud del campo “Source Connection ID”

Destination Connection ID: (0 bit o entre 32 y 144 bits) Indica el número identificador que un emisor asigna a la conexión.

Source Connection ID: (0 bit o entre 32 y 144 bits) Indica el número identificador que el receptor asigna a la conexión.

Cuando un cliente quiere iniciar una conexión, en su primer paquete, al realizar el handshake, establece un número aleatorio que identifique dicha conexión y lo guarda en el campo “Destination Connection ID” (por ejemplo 400). Por tanto, el primer paquete que el cliente envía al servidor tendrá:

Destination Connection ID = 400

Source Connection ID = 0

Al recibir este mensaje, el servidor, copia el valor que ha recibido del cliente y lo establece en su campo “Source Connection ID”. Además, genera otro número aleatorio y lo establece en su campo “Destination Connection ID” (por ejemplo, 600). Por tanto, el primer paquete que el servidor envía al cliente tendrá:

Destination Connection ID = 600

Source Connection ID = 400

El cliente recibe este mensaje, comprueba que el origen tiene el mismo identificador de conexión que él ha establecido (400). Y copia el identificador de conexión destino en su origen. Por tanto, el siguiente paquete que el cliente envía al servidor tendrá:

Destination Connection ID = 400

Source Connection ID = 600

Al recibir este mensaje el servidor comprueba que el origen tiene el mismo ID de conexión que el servidor había establecido. Esto sirve para que tanto servidor como cliente puedan identificar la conexión. En el capítulo 4, sobre seguridad se explica cómo esto supone una forma de mitigar algunos ataques.

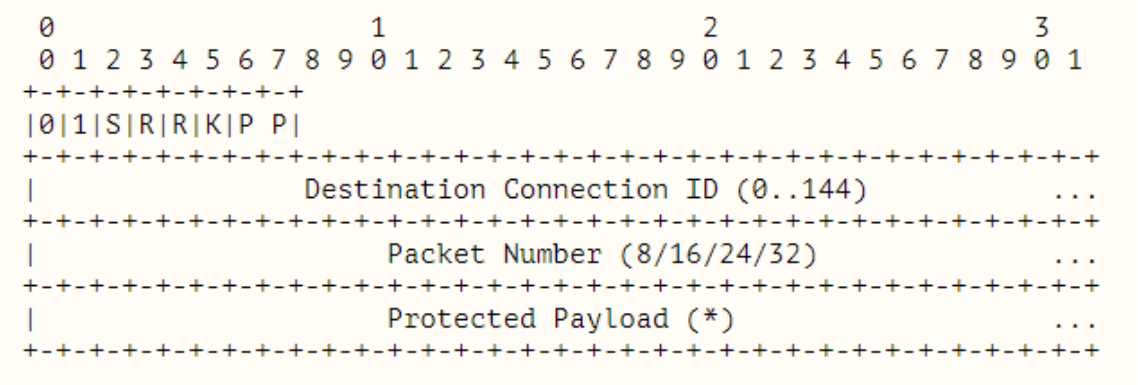


Figura 12: Cabecera corta paquete QUIC. Tomada de [17].

En la Figura 12 puede verse la cabecera corta. Esta cabecera está autenticada pero no encriptada desde el inicio (primeros flags) hasta el número de secuencia. Y a partir el "Packet Number", el resto del paquete se encuentra autenticado y encriptado [17].

Primeros Flags: (8 bits)

- Primer bit: Cuando tiene un 1, indica que se trata de un paquete con cabecera corta.
- Segundo bit: Debe estar siempre activo en esta versión. Los paquetes que no tengan este bit activo se descartarán.
- Tercer bit: Es el bit de SPIN. Cada extremo (cliente y servidor) tienen un valor denominado SPIN que puede ser 0 o 1. Y cuando envían un paquete, establecen ese valor en este bit. Se puede encontrar más información en [18].
- Cuarto y quinto bits: están reservados.
- Sexto bit: indica si permite al destinatario del paquete identificar las claves de protección que se han utilizado para proteger la cabecera (en esta versión de QUIC algunos campos, como el "Packet Number", se encuentran protegidos por un algoritmo de cifrado que utiliza una clave distinta a la de sesión).
- Séptimo y octavo bits: contienen la longitud del número de secuencia del paquete. Representan uno menos de la longitud del campo de número de secuencia en bytes.

Destination Connection ID: es un número aleatorio de hasta 64 bits elegido por el cliente que sirve para identificar la conexión. Si un paquete no tiene este campo se descartará. Esto es importante para mitigar los ataques por denegación de servicio. Se explicará en el capítulo 4, Seguridad en QUIC.

Packet Number: (32 bits) Es el número que le asigna el emisor al paquete. El primer paquete de datos enviado será el 1 y a partir de este el resto de los paquetes serán numerados como uno más el número del paquete previo.

Playload: Después de esta cabecera se encuentran los datos que un terminal quiere enviar al otro. Estos datos están divididos en distintos streams (se explicarán en el punto 6) que se encapsulan dentro del playload (espacio del paquete destinado para almacenar la información).

3.4 ESTRUCTURA PAQUETES QUIC (Google)

En esta sección se explicará la cabecera de los paquetes QUIC según el protocolo desarrollado por Google [19]. En este caso hay una cabecera única (no se diferencia entre una larga y una corta como pasaba con la versión del IETF).

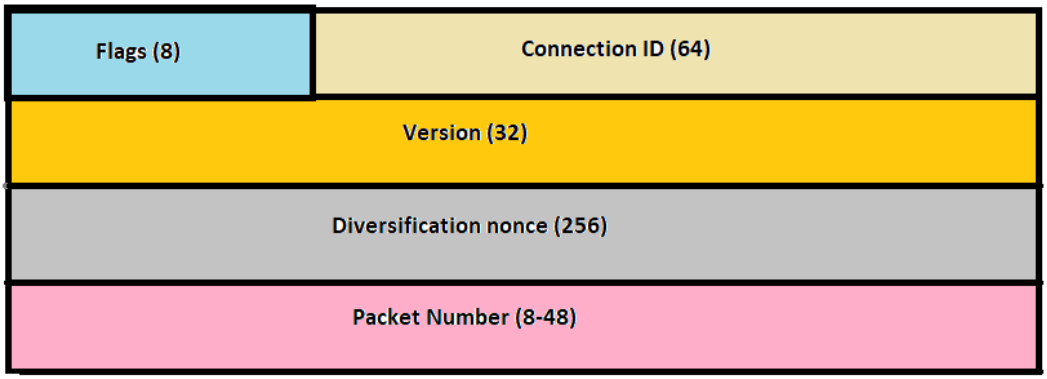


Figura 13: Cabecera paquetes QUIC versión de Google.

Flags: (8 bits)

- Primer bit: La función de este flag varía si el paquete lo envía el cliente o el servidor. Si lo envía el cliente este bit deberá estar activado para indicar que contiene un campo con la versión de QUIC. Sin embargo, el servidor debe enviar los paquetes con este bit a 0, y activarlo sólo cuando quiere cambiar de versión. De modo que cuando un servidor envía un paquete con este bit activado se requiere de una negociación de versión.
- Segundo bit: Se activa únicamente cuando el paquete es un paquete de Reset.

- Tercer bit: Se activa para indicar que la cabecera contiene un campo Diversification nonce.
- Cuarto bit: Se activa para indicar que el paquete contiene un campo Connection ID.
- Quinto y sexto bits: Indican el tamaño del número de paquete.
- Séptimo bit: Este bit está reservado para uso con multipath.
- Octavo bit: Aún no tiene uso, deberá estar a 0.

Connection ID: (0 bit o entre 32 y 144 bits) Indica el número identificador que cliente y servidor asignan a la conexión.

Version: (32 bit) Indica la versión del protocolo que se está utilizando.

Diversification nonce: (256) Este campo solo está presente en los mensajes que el servidor envía al cliente. Es un número que genera el servidor y lo envía en su mensaje Server Hello, sirve para añadir entropía a la generación de claves.

Packet Number: (8-48) Identificador único del paquete.

3.5 ETIQUETAS EN LOS MENSAJES DE QUIC

En la Tabla 1 se muestran las distintas etiquetas que se pueden añadir a los mensajes usados en el establecimiento de conexión para dar seguridad a la comunicación con la versión de QUIC desarrollada por Google [20].

ABREVIATURA	SIGNIFICADO
CHLO	Client Hello
SHLO	Server Hello
SCFG	Server Config
REJ	Reject
CETV	Client encrypted tag – value pairs
PRST	Public Reset
SCUP	Server Config Update
P256	ECDH Curve P256
C255	ECDH Curve 25519
NULL	Null algorithm
AESG	AES128 + GCM-12
CC12	ChaCha20 + Poly1305
STK	Source Address Token

Tabla 1: Etiquetas en los mensajes de QUIC.

Las más importantes son:

CHLO: Consiste en un mensaje que envía el cliente al servidor para indicarle que quiere establecer una conexión con él.

SHLO: Consiste en un mensaje que envía el servidor al cliente para indicarle que quiere establecer una conexión con él.

SCFG: El mensaje "Server Config" contiene la información de la configuración del servidor, como por ejemplo la clave pública del algoritmo de larga duración Diffie-Hellman que se ha utilizado para encriptar el paquete…

STK: Consiste en un bloque encriptado y autenticado que contiene la dirección IP pública del cliente.

REJ: El mensaje "Reject" consiste en un mensaje que envía el servidor al cliente para indicarle que necesita más datos de él. Este mensaje se envía cuando el servidor recibe un mensaje CHLO incompleto (y por tanto no tiene todos los datos que necesita) o cuando recibe un mensaje CHLO completo, pero ya no es válido. Por ejemplo, porque el certificado del servidor ha expirado. Los mensajes Reject contienen entre otros datos, el STK, SCFG, una cadena de confianza con los certificados del servidor, una firma del SCFG utilizando su clave privada…

3.6 DETECTAR QUIC CON HTTPS

Antes de conectarse a un servidor, el cliente no sabe si este servidor es compatible con QUIC o no. Por tanto, realiza una primera petición HTTP mediante el medio tradicional de TLS con TCP, y el servidor contesta a esta petición con una cabecera HTTP (alternative services) que indica que puede utilizar QUIC. Al recibir esta respuesta, el cliente sabe que el servidor también puede utilizar QUIC y a partir de ese momento se utiliza QUIC con ese servidor en lugar de TLS con TCP. [21]

En caso de que la conexión sea demasiado lenta, o el handshake de QUIC falle, se retoma la conexión con TCP y TLS.

3.7 HANDSHAKE (ESTABLECIMIENTO DE CONEXIÓN)

Para comenzar una comunicación entre un cliente y un servidor el primer paso es hacer un handshake. Esto consiste en un intercambio previo de paquetes con el fin de negociar unos parámetros comunes (de seguridad, velocidad, protocolos…) y establecer una conexión entre dos equipos. A diferencia de los protocolos que se explicaron en el capítulo 2, el cliente, con QUIC no espera a recibir la lista de parámetros de seguridad que el servidor soporta, sino que encripta los datos con la versión que él considera (en base a conexiones previas con el servidor) y los envía. Si el servidor entiende el mensaje no hay problema, por el contrario, si el servidor no los entiende, responde al cliente con las versiones que acepta y el cliente vuelve a encriptar el primer mensaje con una de las versiones que ha recibido y lo reenvía al servidor. Este caso causa 1 RTT de retardo al establecimiento de conexión. Esto implica que, si el cliente es capaz de elegir la versión adecuada al principio, QUIC requiere 0 RTT para empezar a enviar datos.

QUIC iniciará el establecimiento de conexión entre el cliente y el servidor de forma diferente según el escenario. No enviará el mismo primer paquete si es la primera vez que trata de conectarse que si ya tiene información almacenada en su caché de una conexión previa con dicho servidor. A continuación, se explican los diferentes primeros paquetes que enviaría el cliente:

Handshake inicial [21]: Se produce cuando el cliente no tiene ninguna información previa sobre el servidor y por tanto necesita averiguar algunos parámetros de este. (Como los que se indican en la Tabla 1).

1) Primero envía un primer paquete denominado "Inchoate CHLO", como se muestra en la Figura 14, este paquete es un client hello incompleto utilizado con el fin de que el servidor nos envíe un Reject con la información que se necesita.
2) El servidor nos envía el paquete Reject, que contiene: un SCFG con la clave pública del algoritmo de larga duración Diffie-Hellman que se ha utilizado para encriptar el paquete, una cadena de confianza que autentifica al servidor, una firma digital del SCFG utilizando la clave privada del certificado de la cadena de confianza y, por último, el STK que contiene la dirección IP del cliente.
3) Una vez recibido este Reject, el cliente comprueba la firma y la cadena de confianza, y si son correctas, envía un Client Hello completo que contiene la clave pública efímera del algoritmo Diffie-Hellman. Además de este CHLO completo, envía también el primer paquete de datos encriptado. Esto es muy importante, ya que, enviando los datos encriptados junto con el CHLO, QUIC puede ahorrarse 1 RTT frente al caso en el que se envían los parámetros de seguridad con intercambio, y después enviar los datos.
4) El servidor, después de recibir esto comprueba que el CHLO sea correcto, y en caso de serlo responde al cliente con un SCHO que contiene la clave pública efímera del algoritmo Diffie-Hellman utilizado. Esto permite que ambos lados puedan calcular las claves de seguridad finales. El servidor, envía además después de este paquete, el ACK correspondiente al primer paquete de datos que ha recibido.

Esto implica que QUIC utiliza dos niveles de seguridad. Un primer nivel se lo proporciona el hecho de encriptar los datos del cliente con sus claves iniciales (esta seguridad es similar a la que proporciona el protocolo TLS). El segundo nivel de seguridad es debido al uso del algoritmo Diffie-Hellman efímero que negocian cliente y servidor. Y esta

seguridad es aún mayor que la del primer nivel, ya que expira al cabo de poco tiempo y en el caso de que un posible atacante se haga con la clave, seguramente ya no podrá utilizarla porque habrá expirado y se ha generado una nueva.

En conclusión, como se muestra en la Figura 14, este establecimiento de conexión tarda 1 RTT, que ocurre desde que el cliente envía el CHLO incompleto hasta que recibe el Reject por parte del servidor. Aunque la conexión no está completamente establecida, en el próximo paso el cliente envía ya datos al servidor, por tanto, sólo ha tardado 1 RTT en establecer una conexión que le permita enviar datos.

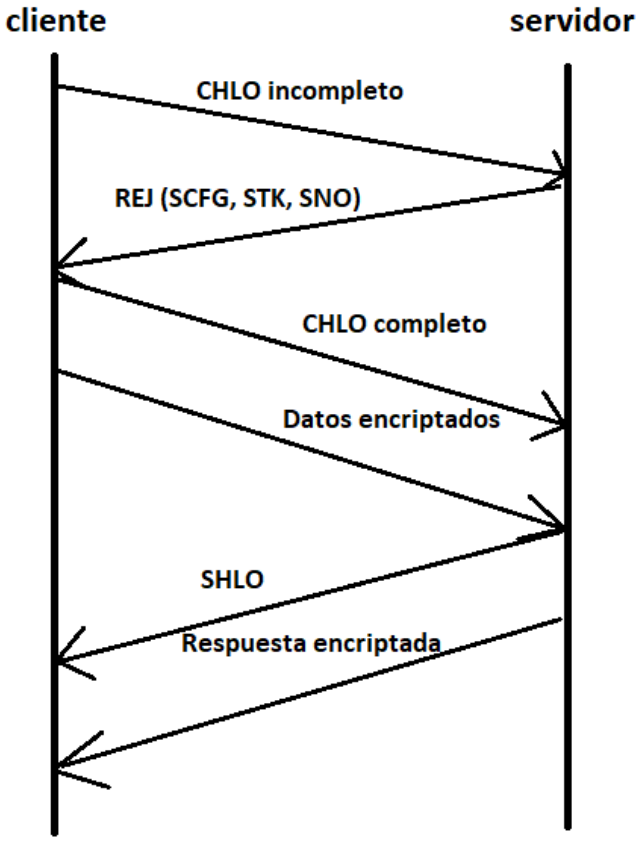


Figura 14: Establecimiento de conexión 1 RTT.

Handshake retomando una conexión previa [21]: El cliente guarda en su cache los valores del SCFG y STK que ha recibido del servidor en una conexión anterior que se ha llevado a cabo hace poco tiempo. Esto le permite tener la información necesaria del servidor para poder retomar una conexión con él sin necesidad de enviarle un mensaje incompleto para que éste le responda con un mensaje Reject que contenga estos datos. Esto se muestra en la Figura 15.

1) Primero el cliente envía un CHLO completo junto con el primer paquete de datos encriptados.
2) Y si ha habido éxito en este primer envío, el servidor responde con un SHLO junto con el ACK correspondiente a este primer paquete.

Gracias a almacenar esta información del servidor, el cliente consigue enviar un paquete de datos en su primer intercambio de paquetes con el servidor. Por tanto, en esta situación, QUIC requiere de 0 RTT para enviar datos.

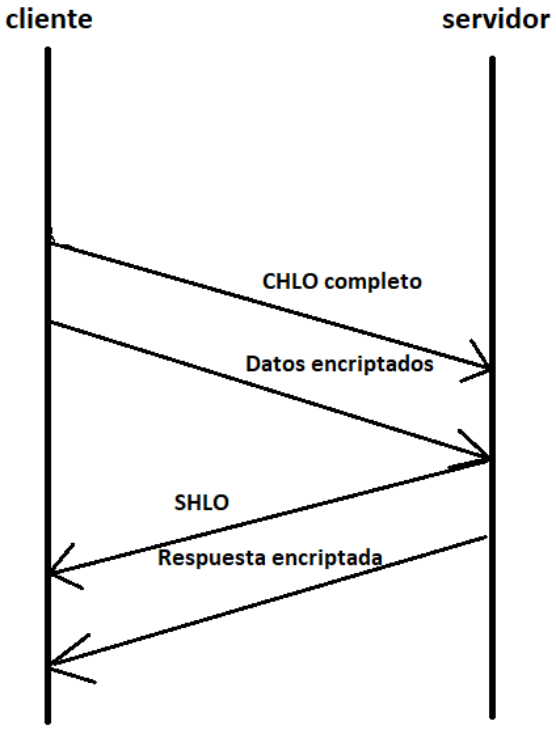


Figura 15: Establecimiento de conexión 0 RTT.

Handshake retomando una conexión previa que ha caducado [21]: Es posible que el cliente tenga almacenados el SCFG y STK del servidor de una conexión previa que tuvo lugar hace poco tiempo y que el STK haya expirado o el servidor haya cambiado de certificado. Esto implicaría que el cliente intentaría retomar la conexión como en el caso anterior, Figura 15. Sin embargo, el servidor ya no puede reconocer al cliente debido a su nuevo certificado o STK.

1) Primero, el cliente envía el mensaje “CHLO completo” junto con el primer paquete de datos encriptados.

2) Esta vez el servidor no acepta este mensaje. Entonces, el servidor necesita enviar un mensaje Reject al cliente indicándole los nuevos datos del certificado y el STK.
3) El cliente, una vez recibe este mensaje, elabora un nuevo mensaje CHLO completo y lo envía. A continuación, envía un nuevo paquete que contiene los datos encriptados que se enviaron anteriormente pero el servidor no aceptó.
4) Al recibir esto el servidor, si la información del CHLO es correcta, responde con un SHLO y el ACK del paquete de datos que ha recibido y ahora si desencripta los datos que ha recibido y establece la conexión.

En conclusión, aunque haya expirado el token o el servidor cambie de certificados, QUIC necesitará enviar un mensaje más al servidor, pero debido a que junto con este mensaje envía datos, ahora el tiempo sería de 1 RTT necesario hasta que envía el primer paquete de datos encriptados al servidor.

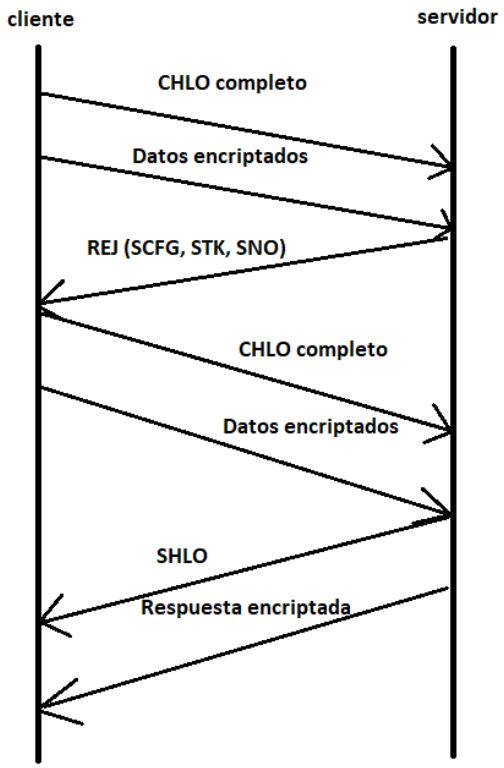


Figura 16: Establecimiento conexión previa con sesión caducada (1 RTT).

3.8 COMPARACIÓN DEL TIEMPO ENTRE PROTOCOLOS

En los capítulos anteriores se han tratado las distintas formas de establecer conexiones entre cliente y servidor que realiza cada protocolo. La Tabla 2 recoge el tiempo necesario

para descargar, desde una web, 3 objetos sin ninguna conexión previa, suponiendo que son objetos muy pequeños cuyo contenido puede enviarse en un único paquete:

PROTOCOLO	TIEMPO (RTT)
HTTP/1.0 con TCP y TLS	12
HTTP/1.1 con TCP persistente y TLS	6
HTTP/1.1 con TCP persistente y TLS 1.3	5
HTTP/2 con TCP persistente y TLS 1.3	3
QUIC	2

TABLA 2: Comparación tiempo entre protocolos sin conexión previa

En la Tabla 3 se muestran las mismas comparaciones, pero teniendo en cuenta que se ha establecido una conexión previa entre ambos hace poco tiempo, suponiendo que son objetos muy pequeños que requieren únicamente de 1 RTT cada uno:

PROTOCOLO	TIEMPO (RTT)
HTTP/1.0 con TCP y TLS	12
HTTP/1.1 con TCP persistente y TLS	6
HTTP/1.1 con TCP persistente y TLS 1.3	4
HTTP/2 con TCP persistente y TLS 1.3	2
QUIC	1

Tabla 3: Comparación tiempo entre protocolos con conexión previa

Como se puede ver la diferencia entre el tiempo que se tarda en enviar datos entre QUIC y las primeras versiones de HTTP con TLS es muy grande. Sin embargo, cabe notar que este ejemplo se ha propuesto con una página que tiene 3 objetos. Si se compara con una página web estándar en la actualidad, que rondan los 100 objetos, la diferencia es mucho mayor.

3.9 STREAMS

En este apartado se describe cómo, el protoclo QUIC, gestiona el envío de contenidos en paralelo mediante el uso de streams.

3.9.1 MULTIPLEXACIÓN DE STREAMS

El protocolo QUIC se basa en el uso de streams diferentes para gestionar la descarga de cada contenido. De esta manera, después de la cabecera de los paquetes, se envía la información separada en diferentes streams, que pueden tener un tamaño variable según las necesidades de la comunicación. Un único stream puede ocupar todo el paquete si no se necesita transmitir por otros streams. Sin embargo, la intención de QUIC es que los streams sean pequeños. Esto crea la necesidad de un nuevo identificador, distinto al identificador del paquete, que diferencie los streams.

Un stream puede ser unidireccional o bidireccional, según permita tráfico en un sentido o en ambos sentidos entre dos equipos. Por tanto, hay 4 tipos de streams: unidireccional proveniente del cliente, unidireccional proveniente del servidor, bidireccional proveniente del cliente o bidireccional proveniente del servidor.

Llevar a cabo un intercambio de datos a través de un stream bidireccional es posible gracias a un sistema de estados. En el cual se establece un estado de escucha y otro de transmisión en un mismo stream para evitar colisiones. [21]

Un servidor, para diferenciar los streams que recibe necesita un indicador de inicio y fin de cada stream que va dentro del paquete. El inicio no se indica de ninguna forma, sino que corresponde con el primer bit del stream, mientras que el final del stream se indica con un bit de “FIN” al final de los datos. Y con la ayuda del offset y el ID del stream, el equipo es capaz de ordenar los datos que ha recibido.

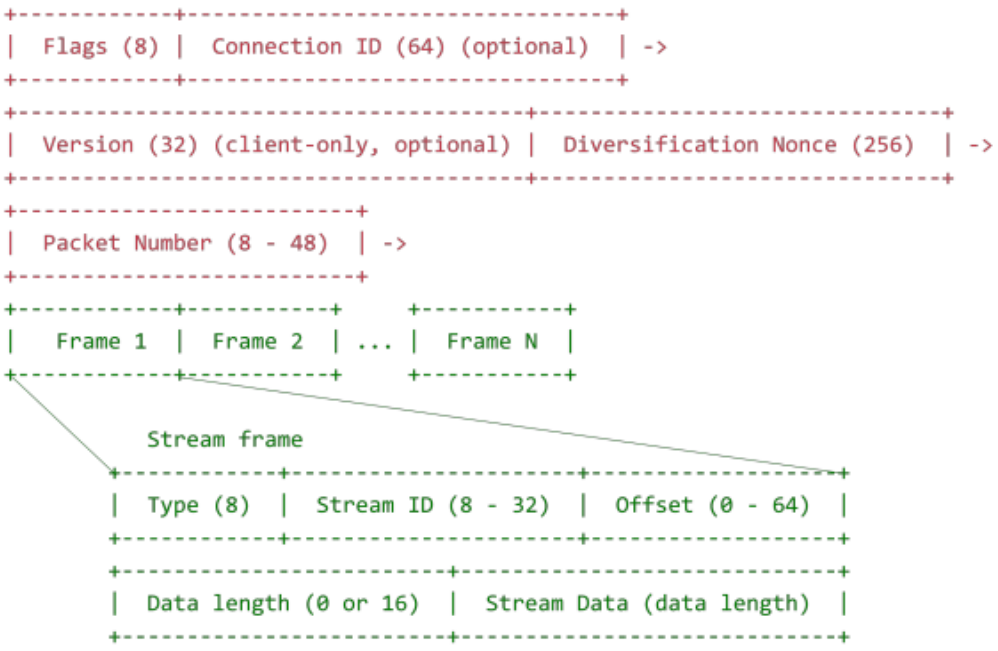


Figura 17: Cabecera QUIC y playload con la estructura de los frame de streams. Tomada de [21].

Como se puede ver en la Figura 17, los paquetes QUIC utilizados para el tráfico de información se componen de: una cabecera como la vista en la sección 3.4 a la que se le añade la información multiplexada de los diferentes streams. Esta información se separa en diferentes estructuras o "Frames" que se componen de [21]:

Tipo: (1 byte) indica el tipo de frame del stream.

Stream ID: (de 1 a 4 bytes) es un número único que identifica a cada stream y, por tanto, no se debe repetir para otro stream en la misma conexión. Se ha establecido un identificador que será un número par si el stream procede del cliente, o un número impar, si por el contrario el stream procede del servidor. No obstante, el primer stream de una conexión tendrá el identificador 0.

Tamaño de datos: (de 0 a 2 bytes) indica el tamaño del campo que contiene los datos que se desean enviar.

Datos: (tantos bytes como indique el campo "Tamaño de datos") es la información que un equipo envía al otro.

3.9.2 PRIORIDAD DE INFORMACIÓN EN QUIC

El protocolo QUIC no tiene una estructura o un campo específico diseñados para diferenciar prioridades entre streams, trata a todos por igual. A pesar de ello, puede surgir la necesidad de enviar una información antes que otra y por tanto distinguir prioridades. La solución consiste en que sea la aplicación la que establezca dichas prioridades y envíe primero la información con prioridad más alta, guardando en su buffer la de menor prioridad para enviarla cuando sea posible [21].

3.9.3 CANCELACIÓN DE UN STREAM

Una vez el cliente, o el servidor, ha decidido que ya no se necesita más información proveniente de un stream, es posible cancelar dicho stream sin cortar toda la conexión entre el cliente y el servidor. Por tanto, el resto de streams pueden seguir intercambiando información. [21].

3.9.4 CONCLUSIÓN SOBRE EL USO DE STREAMS

Una vez se ha terminado de enviar información por medio de un stream, es posible cancelar dicho stream sin cancelar toda la conexión entre los extremos. Esto implica que el acceso a gran cantidad de páginas o aplicaciones web sea más rápido. Actualmente las páginas web no se componen solo de un texto, sino que tienen imágenes, vídeos, anuncios, y gran cantidad de elementos web.

Sin embargo, con la llegada de este protocolo, estas consultas web pueden realizarse estableciendo una única conexión y obteniendo la información necesaria para las distintas aplicaciones web por medio de streams que pueden ser cancelados una vez han dejado de utilizarse. Realizar una única conexión implica menos handshakes y menos retardos por establecimientos de conexión y, por tanto, se puede acceder a la información más rápido.

QUIC utiliza streams que son multiplexados en una sola conexión. Así la pérdida de un paquete sólo bloquea el contenido del stream correspondiente a ese paquete. Gracias a esto se elimina retardos por “Head of Line Blocking”.

3.10 RECUPERACIÓN FRENTE A PÉRDIDAS

En TCP cuando un paquete no llega a su destino en un intervalo de tiempo razonable, se reenvía dicho paquete con el mismo número identificador. Esto puede provocar un problema de ambigüedad ya que al recibir el ACK el cliente no sabe si corresponde al paquete original o al paquete reenviado. Esto no supone un problema importante para el transcurso de la comunicación, pero el diferenciar los paquetes facilita la detección de pérdidas o congestión. Es por eso que en QUIC se utilizan identificadores únicos incluso en los paquetes que se reenvían.

Además, QUIC añade en los ACK el tiempo que transcurre desde que ha recibido el paquete hasta que envía su correspondiente ACK. Esto permite al cliente calcular el RTT de una conexión de manera más precisa y facilita la detección de un problema en la comunicación que implique la pérdida de paquetes. [21]

3.11 CONTROL DE FLUJO

En una comunicación entre un cliente y un servidor, ambos equipos tienen un buffer para almacenar los datos de los paquetes que han recibido y procesarlos mientras que el resto de los paquetes con datos siguen viajando. El número de paquetes que puede enviar un equipo al otro sin que su tamaño supere el que le permite el buffer del destinatario se conoce como ventana. En QUIC se diferencian dos ventanas; la ventana del stream y la ventana de la comunicación. La ventana de la comunicación hace referencia al número de paquetes que se podrán enviar entre todos los stream sin sobrepasar el buffer del receptor, mientras que la ventana del stream se refiere al número de paquetes que puede enviar ese stream concreto para no sobrepasar la cantidad que el buffer del receptor le ha reservado. Por tanto, la ventana de la comunicación será mayor o igual (solo en el caso de que se transmita un único stream) que la ventana de stream. Ambas ventanas se calculan de la misma forma, tal y como se explica a continuación.

Cuando se establece una conexión entre dos equipos, el receptor guarda la información pendiente de ser leída por la aplicación en buffers. Esto podría producir que, si la aplicación receptora procesa muy lentamente la información de un stream, el buffer se llenará con la información de este stream y se podría producir “Head of Line blocking” y no sería capaz de procesar la información que le llega de otros streams. Para solucionar

esto, QUIC limita el tamaño que el buffer debe reservar para cada stream, siendo este el tamaño de la ventana de stream.

Para calcular este tamaño que debe reservar en el buffer a cada stream, considerará las necesidades de la comunicación en ese momento. Primero, el receptor indica al emisor cuanto está dispuesto a reservar, y periódicamente, con los ACKs que el servidor envía al cliente para confirmar la recepción de paquetes, el receptor puede enviar actualizaciones que indiquen al emisor que está dispuesto a aumentar el tamaño que le reserva del buffer y de esta forma el emisor puede enviar mayor cantidad de datos. [21]

3.12 CONTROL DE CONGESTIÓN

QUIC utiliza el mismo mecanismo de control de congestión que muchas de las implementaciones actuales de TCP. Este mecanismo se conoce como CUBIC, llamado así por calcular el tamaño de la ventana por medio de una función cúbica del tiempo que ha transcurrido desde el último evento que produjo congestión en la comunicación. En la Figura 18 se puede ver la forma de una función cúbica.

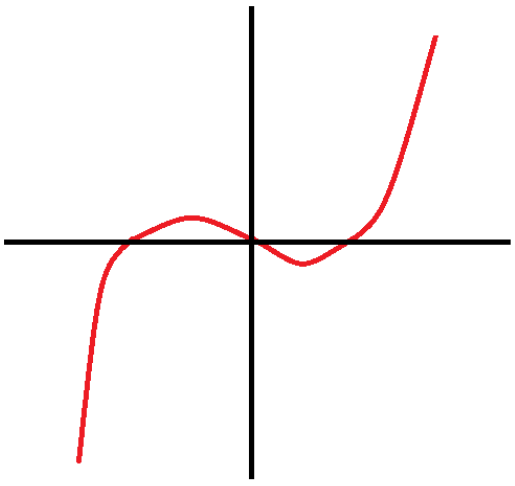


Figura 18: Función cúbica.

Como se ve en la función, hay un primer momento en el que la ventana aumentará rápidamente hasta llegar al primer punto de inflexión (que será el tamaño que tenía la ventana antes de la congestión). Después de alcanzar este tamaño la ventana estará

bastante tiempo entre los dos puntos de inflexión buscando encontrar el equilibrio y estabilizarse, una vez se ha estabilizado crece rápidamente de nuevo. Por tanto, CUBIC se caracteriza porque tiene dos períodos de crecimiento rápido de la ventana. Uno cóncavo y otro convexo. Y entre estos periodos de rápido crecimiento hay un periodo de estabilización. CUBIC además se caracteriza porque solo tiene en cuenta el tamaño previo de la ventana, y no tienen en cuenta otros aspectos como el RTT, que sí tienen en cuenta otros algoritmos de control de congestión.

3.13 CONLUSIONES DEL CAPÍTULO

Hay dos versiones principales del protocolo QUIC, la de Google y la del IETF. Ambas son muy similares, debido a que Google también está interesado en la estandarización del protocolo y colabora, junto con otras empresas, en el proceso de estandarización en el IETF. Por otro lado, Google también actualiza su protocolo para asemejarlo a la versión que se está estandarizando en el IETF.

En cuanto a los paquetes, son muy similares. Cabe destacar que la cabecera de los paquetes de Google junta las cabeceras larga y corta del IETF, teniendo unos flags similares, un campo con la versión, otro con el número del paquete, un campo llamado “diversification nonce” que es opcional, y la mayor diferencia, es que no tiene un espacio reservado para indicar la longitud del ID de conexión y sólo utiliza un ID para la conexión que comparten cliente y servidor, mientras que en la versión del IETF se diferencia entre una conexión para el cliente y otro número distinto para el servidor.

QUIC utiliza un sistema de handshake que guarda en su memoria caché datos de conexiones previas con ese servidor. Esto le permite establecer una conexión y enviar datos en 0RTT en el mejor de los escenarios, en lugar de los 1RTT que se tarda con HTTP/2 con TCP persistente y TLS1.3.

El protocolo QUIC tiene sistemas de control frente a pérdidas y controles de flujo y congestión. Estos mecanismos permiten llevar a cabo conexiones estables.

4. SEGURIDAD EN QUIC

En este capítulo se describen algunos de los ataques que se podrían llevar a cabo para tratar de romper la confidencialidad o integridad de las comunicaciones y cómo QUIC consigue evitar o mitigar dichos ataques.

4.1 CRIPTOGRAFÍA EN LOS HANDSHAKE

Como se ha expuesto en el apartado 3.7 QUIC otorga dos niveles de seguridad a sus conexiones. El primero consiste en enviar un primer mensaje encriptado suponiendo que el servidor entenderá dicho mensaje. En el caso de que no lo entienda, responderá con los posibles algoritmos que puede utilizar para encriptarlos. así, el cliente volvería a enviar los datos encriptados con uno de esos algoritmos que soporta el servidor. El segundo nivel viene dado por una clave efímera Diffie-Hellman.

Durante el handshake se deberá llevar a cabo un intercambio de claves que proporcione los siguientes servicios de seguridad [22]:

- El servidor deberá estar siempre autenticado.

- Será opcional que el cliente esté autenticado.

- Para cada conexión, se deberán generar claves distintas que no estén relacionadas entre sí.

- El uso de claves para encriptar los paquetes se debe usar tanto en el caso de que el establecimiento de conexión requiera de 1 RTT como en el caso de que requiera 0 RTT

4.2 DENEGACIÓN DE SERVICIO

Se ha explicado que QUIC es un protocolo basado en la encriptación y autenticación. Por tanto, tiene mecanismos de seguridad frente a ataques por DDoS (denegación de servicio, consiste en hacer que los recursos de un servidor sean inaccesibles para el resto de usuarios, generalmente se provoca enviando muchas peticiones de conexión al servidor, saturando así su ancho de banda o capacidad de procesamiento).

Cuando uno de los extremos que detecta que un paquete que ha recibido no está autenticado lo descarta (excepto algunos paquetes ICMP, aunque su uso es muy limitado). Esto evita que un posible atacante interfiera con conexiones ya establecidas.

Otro mecanismo de defensa que tiene QUIC frente a estos ataques es su identificador de conexión. (en el capítulo 3 se explicó esta etiqueta). Este identificador único será un número aleatorio e impredecible elegido por el cliente y conocido tanto por el cliente como por el servidor. Y un posible ataque que pretenda interceder en la conexión, tendrá un número de conexión diferente. Al detectar que el número no coincide, el servidor lo descarta [17].

Es importante destacar que estos mecanismos de seguridad están destinados a proteger conexiones activas, ya que, como se ha explicado en el punto anterior es en el handshake donde se realiza la protección frente a nuevas conexiones con equipos no deseados.

4.3 ATAQUE MEDIANTE REUTILIZACIÓN DE STK

Se ha explicado en el punto 3.5 que las cabeceras de QUIC tienen un campo STK. Un atacante podría obtener dicho campo y conseguir la dirección IP del cliente y por tanto tendría la información necesaria para intentar una conexión 0 RTT de la cual recibiría un primer paquete con datos.

La solución a este posible ataque es limitar el tiempo de validez del STK para establecer una conexión 0 RTT. Y esto implicaría que al expirar un STK no se puede llevar a cabo una conexión 0 RTT y tendría que ser una conexión 1 RTT como se explica en el punto 3.7 [17]

4.4 ATAQUE SLOWLORIS

Este ataque consiste en un tipo específico de ataque por denegación de servicio. Lo que lo caracteriza es que una sola máquina puede desbordar un servidor. Para ello se envían peticiones de conexión, y una vez abierta la conexión se trata de mantenerla abierta el mayor tiempo posible para consumir sus recursos. El atacante envía un paquete pequeño cada cierto tiempo con la única intención de mantener la conexión abierta y que no se cierre por inactividad. De esta forma se puede simular que la red está bajo a un problema de pérdidas de paquetes.

Hay varios mecanismos que solucionarían este problema, como disminuir el tiempo que una conexión puede permanecer abierta sin que se estén enviando datos, limitar el número de conexiones que puede realizar una máquina con una misma dirección IP, o aumentar la velocidad mínima que debe tener una conexión [17].

4.5 ATAQUES POR FRAGMENTACIÓN DE STREAMS

Este ataque se puede llevar a cabo de dos maneras, la primera consiste en enviar fragmentos de streams con el fin de provocar en el receptor, que los buffers de almacenamiento tengan que reservar un espacio desmesurado para algunos streams.

Otra manera de realizar este tipo de ataque, consiste en no enviar deliberadamente los ACK de los stream que se reciben del servidor. Esto provoca que el servidor tenga que guardar en su buffer la información de la cual no ha recibido el correspondiente ACK.

Este ataque sería mitigado por la ventana de control de flujo explicado en el punto 3.10. [17].

4.6 ATAQUES POR DEGRADACIÓN DE VERSION

Anteriormente se explicaron las distintas versiones de QUIC.

Debido a que la versión que se utilizará del protocolo se negocia en el handshake, es posible que un atacante solicite llevar a cabo una conexión en una versión más antigua donde no haya solución a algún posible ataque.

De momento no hay un método efectivo para evitar este tipo de ataque. Por tanto, en futuras versiones de QUIC se espera incluir un sistema que sea robusto frente a este tipo de ataques [17].

4.7 INTERCAMBIOS DE PARÁMETROS DE SEGURIDAD

En el punto 3.7 se explicó cómo el cliente y el servidor, en el handshake se intercambian distintos mensajes con parámetros. En concreto el SCFG que contiene el certificado X.509 del servidor y sus claves públicas que permiten el uso de curvas elípticas con el algoritmo Diffie-Hellman. El servidor también puede utilizar el algoritmo de cifrado RSA, y enviar su certificado firmado con dicho algoritmo.

En QUIC es únicamente el servidor quien debe firmar su certificado. Mientras que la autenticación del cliente es opcional. También es importante notar, como se explicó en el punto 3.7, que solo se realiza esta autenticación en el caso de handshake de 1 RTT (no ha habido una conexión previa reciente entre cliente y servidor) y por tanto en el caso de que haya habido una conexión previa reciente el servidor no enviará un mensaje SCFG y no se autenticará de nuevo.

El documento [22] muestra más información sobre cómo llevar a cabo un ataque a QUIC o TLS utilizando los métodos de Manger o Bleichenbacher.

4.8 CONCLUSIONES DEL CAPÍTULO

En este capítulo se han visto algunos de los ataques más comunes llevados a cabo para tratar de romper la seguridad de una conexión entre un cliente y un servidor. QUIC cuenta con diferentes mecanismos que evitan o mitigan estos ataques, convirtiéndolo en un protocolo igual o más seguro que sus predecesores.

5. EVALUACIÓN DE QUIC

En este capítulo se evalúa el uso QUIC en Internet. Para ello, primero se describe brevemente la herramienta utilizada para analizar el tráfico web, Wireshark.

Como se ha explicado antes, hay dos versiones de QUIC, y cada versión es tratada como un protocolo diferente en Wireshark, que les da nombres diferentes, siendo QUIC los paquetes de la versión del IETF y GQUIC los de la de Google. En este capítulo se evalúan ambas versiones, aunque la única versión con un despliegue significativo actualmente es la de Google.

Se evalúa la compatibilidad de ambas versiones con varios navegadores web, sitios web, redes de distribución de contenido, software de servidores web y se estudia la estructura de GQUIC para ver si coincide con la teoría del capítulo 3.

5.1 HERRAMIENTA PARA CAPTURAR TRÁFICO: WIRESHARK

Para llevar a cabo muchas de las pruebas de este capítulo se ha utilizado el analizador de protocolos Wireshark. Wireshark permite capturar el tráfico web generado y recibido por una interfaz de nuestro ordenador y analizar los diferentes protocolos que han intervenido en la comunicación, así como sus etiquetas y su contenido.

Las versiones más modernas de Wireshark, como la 3.0.1 y superiores, reconocen la estructura de los protocolos GQUIC (protocolo QUIC de Google) y QUIC (protocolo QUIC del IETF), por tanto, al igual que se puede filtrar paquetes UDP, TCP o HTTP escribiendo el nombre del protocolo en la barra de filtrado, también se pueden filtrar paquetes GQUIC y QUIC. Sin embargo, como este protocolo está en constante evolución es necesario actualizar Wireshark frecuentemente para seguir los cambios que se están produciendo en ambas versiones del protocolo. De hecho, como se verá más adelante, la versión más reciente de GQUIC (46) no se puede decodificar con Wireshark.

Dentro del programa Wireshark, en el apartado “Expressions” se encuentran los distintos protocolos y campos por los que se puede filtrar el tráfico capturado. Si buscamos el protocolo GQUIC, se puede ver un desplegable en el que se muestran otros posibles filtros

que se pueden aplicar a este tipo de paquetes. Algunos de los campos por los que se pueden filtrar los paquetes en GQUIC son: gquic.cid para filtrar por su "Connection ID", gquic.data_len para filtrar por la longitud del paquete, gquic.version para filtrar por la versión…

Lo mismo ocurre para la versión de QUIC del IETF, en este caso, habrá que buscar el desplegable llamado QUIC, en el que se encuentran otros campos como quic.dcid para filtrar por el "Connection ID" de destino, quic.dcidl para filtrar por la longitud del "Connection ID" de destino, quic.version para filtrar por la versión del protocolo…

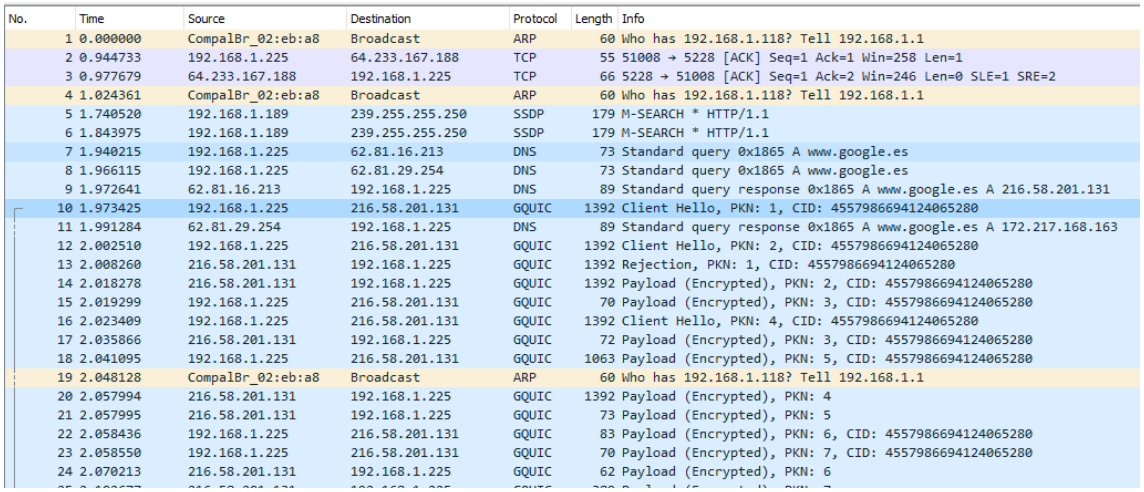
La forma de aplicar filtros sobre estos campos es escribir el nombre del campo por el que se quiere filtrar, y añadir alguno de los siguientes símbolos "==", "<", ">", "<=" … según si quiere filtrarse para obtener los paquetes que tengan un determinado campo igual a un valor, menor que dicho valor, mayor, menor o igual…

5.2 COMPATIBILIDAD DE QUIC CON NAVEGADORES WEB

En la actualidad, los navegadores web más utilizados son Google Chrome, Mozilla Firefox, Microsoft Edge, Opera y Safari. En este apartado se va a estudiar su compatibilidad o no con la versión de QUIC de Google (GQUIC) y la del IETF (QUIC).

- Mozilla Firefox: Este navegador no soporta QUIC ni GQUIC en la actualidad. Su versión estable más reciente es la 67.0.2.
- Microsoft Edge: Por el momento Microsoft Edge no soporta el protocolo QUIC ni GQUIC. La versión más reciente de este navegador es la 44.17763.1.0.
- Opera: Este navegador no soporta QUIC. Sin embargo, desde la versión Opera 16, soporta el protocolo GQUIC. En este capítulo se ha usado la versión 60.0.3255.151 del navegador. La forma de activar el uso del protocolo GQUIC es mediante la URL "opera://flags/#enable-quic". Se ha descargado este navegador y se ha comprobado que la configuración inicial de este flag es "por defecto", por lo que es necesario activarlo para poder usar GQUIC al conectarse a una web que lo permita.

  En la Figura 19 se muestra una captura de tráfico tomada con Wireshark al acceder a Google.com mediante el navegador Opera. Como puede verse, se produce tráfico GQUIC.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	CompalBr_02:eb:a8	Broadcast	ARP	60	Who has 192.168.1.118? Tell 192.168.1.1
2	0.944733	192.168.1.225	64.233.167.188	TCP	55	51008 → 5228 [ACK] Seq=1 Ack=1 Win=258 Len=1
3	0.977679	64.233.167.188	192.168.1.225	TCP	66	5228 → 51008 [ACK] Seq=1 Ack=2 Win=246 Len=0 SLE=1 SRE=2
4	1.024361	CompalBr_02:eb:a8	Broadcast	ARP	60	Who has 192.168.1.118? Tell 192.168.1.1
5	1.740520	192.168.1.189	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
6	1.843975	192.168.1.189	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
7	1.940215	192.168.1.225	62.81.16.213	DNS	73	Standard query 0x1865 A www.google.es
8	1.966115	192.168.1.225	62.81.29.254	DNS	73	Standard query 0x1865 A www.google.es
9	1.972641	62.81.16.213	192.168.1.225	DNS	89	Standard query response 0x1865 A www.google.es A 216.58.201.131
10	1.973425	192.168.1.225	216.58.201.131	GQUIC	1392	Client Hello, PKN: 1, CID: 4557986694124065280
11	1.991284	62.81.29.254	192.168.1.225	DNS	89	Standard query response 0x1865 A www.google.es A 172.217.168.163
12	2.002510	192.168.1.225	216.58.201.131	GQUIC	1392	Client Hello, PKN: 2, CID: 4557986694124065280
13	2.008260	216.58.201.131	192.168.1.225	GQUIC	1392	Rejection, PKN: 1, CID: 4557986694124065280
14	2.018278	216.58.201.131	192.168.1.225	GQUIC	1392	Payload (Encrypted), PKN: 2, CID: 4557986694124065280
15	2.019299	192.168.1.225	216.58.201.131	GQUIC	70	Payload (Encrypted), PKN: 3, CID: 4557986694124065280
16	2.023409	192.168.1.225	216.58.201.131	GQUIC	1392	Client Hello, PKN: 4, CID: 4557986694124065280
17	2.035866	216.58.201.131	192.168.1.225	GQUIC	72	Payload (Encrypted), PKN: 3, CID: 4557986694124065280
18	2.041095	192.168.1.225	216.58.201.131	GQUIC	1063	Payload (Encrypted), PKN: 5, CID: 4557986694124065280
19	2.048128	CompalBr_02:eb:a8	Broadcast	ARP	60	Who has 192.168.1.118? Tell 192.168.1.1
20	2.057994	216.58.201.131	192.168.1.225	GQUIC	1392	Payload (Encrypted), PKN: 4
21	2.057995	216.58.201.131	192.168.1.225	GQUIC	73	Payload (Encrypted), PKN: 5
22	2.058436	192.168.1.225	216.58.201.131	GQUIC	83	Payload (Encrypted), PKN: 6, CID: 4557986694124065280
23	2.058550	192.168.1.225	216.58.201.131	GQUIC	70	Payload (Encrypted), PKN: 7, CID: 4557986694124065280
24	2.070213	216.58.201.131	192.168.1.225	GQUIC	62	Payload (Encrypted), PKN: 6

Figura 19: Tráfico GQUIC con Opera.

- Safari: Actualmente este navegador no soporta QUIC ni GQUIC [23]. La última versión estable del navegador es la 12.0
- Google Chrome: Como ya se ha mencionado anteriormente, este navegador es compatible con GQUIC, pero no con QUIC. Para este capítulo se ha utilizado la versión 75.0.3770.80. Chrome nos da la posibilidad de habilitar o deshabilitar su uso. Hay varias formas de hacerlo. La más sencilla es ir a la URL "chrome://flags", buscar la opción "Experimental QUIC Protocol" y deshabilitarla. De esta manera las consultas se harán por medio de HTTP y TLS sobre TCP. Si en lugar de deshabilitarlo se habilita, o se deja por defecto, el tráfico se servirá por GQUIC si es posible.

  Se ha comprobado el funcionamiento del protocolo GQUIC con los diferentes valores de este flag. En los casos en que está deshabilitado nunca se producen conexiones utilizando GQUIC. Por el contrario, cuando el flag se encuentra con el valor "enabled" (habilitado), todas las conexiones entre Chrome y servidores de Google o páginas pequeñas que también soportan el protocolo (como https://www.accelerate.education) se llevan a cabo mediante la versión 43 del protocolo GQUIC (esta versión se ha comprobado como se indica en el capítulo 5.2.2). Por último, para el valor "default" (por defecto) del flag QUIC de Chrome, se ha vuelto a establecer conexión con las mismas páginas y se ha comprobado como se indica en 5.2.2 la versión del protocolo. En los casos en que se ha

conectado a servidores de Google, la conexión se ha llevado a cabo mediante la versión 46 de GQUIC, y en los casos en que se ha llevado a cabo una comunicación con páginas pequeñas como https://www.accelerate.education no ha sido posible utilizar esta versión del protocolo y se ha llevado a cabo mediante HTTP/TCP/TLS.

Además, se ha capturado el tráfico en estos casos con Wireshark. Para las comunicaciones con el flag de GQUIC en habilitado se produce tráfico GQUIC, sin embargo, para las comunicaciones con servidores de Google con el flag "default" se produce tráfico UDP en el puerto 443 pero Wireshark no lo reconoce como paquetes GQUIC. Todo esto quiere decir que, al usar Chrome con el flag por defecto, se intenta establecer conexión con la versión de GQUIC más reciente y los servidores más pequeños aún no la soportan y la herramienta Wireshark tampoco reconoce aún la versión más moderna de GQUIC.

5.2.1 REGISTROS DE RED EN GOOGLE CHROME

Con el fin de depurar posibles problemas en la red, el navegador Chrome cuenta con una herramienta que guarda los registros de red en nuestro equipo en un fichero del tipo ".json". Además, este navegador tiene otra herramienta que nos permite visualizar este tipo de ficheros para analizar los estados de nuestras comunicaciones.

Anteriormente, estas herramientas se podían utilizar accediendo a la URL "chrome://net-internals/#events". Sin embargo, ahora Chrome cuenta con una nueva herramienta más completa. En primer lugar, es necesario crear un registro log, para ello basta con acceder a la URL "chrome://net-export", elegir si queremos que Chrome incluya las credenciales y cookies o datos crudos en el fichero y por último pulsar el botón "Start Logging to Disk" (Figura 20). Al hacer esto, aparecerá una nueva ventana en la que se puede elegir el nombre y la ubicación del fichero que Chrome generará.

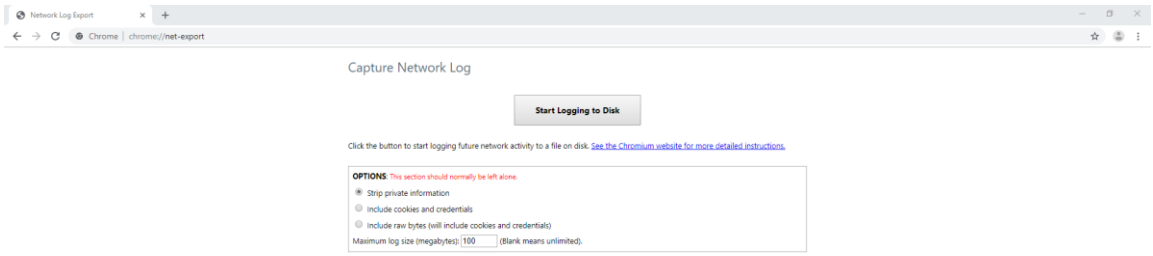


Figura 20: Crear registro de red con Chrome.

Una vez se ha creado este registro, se puede visualizar su contenido mediante la URL "https://netlog-viewer.appspot.com/#import". Para ello, basta con importar el fichero. Esta herramienta nos permite analizar varios aspectos de las conexiones (mostradas en la barra de navegación de la Figura 21).

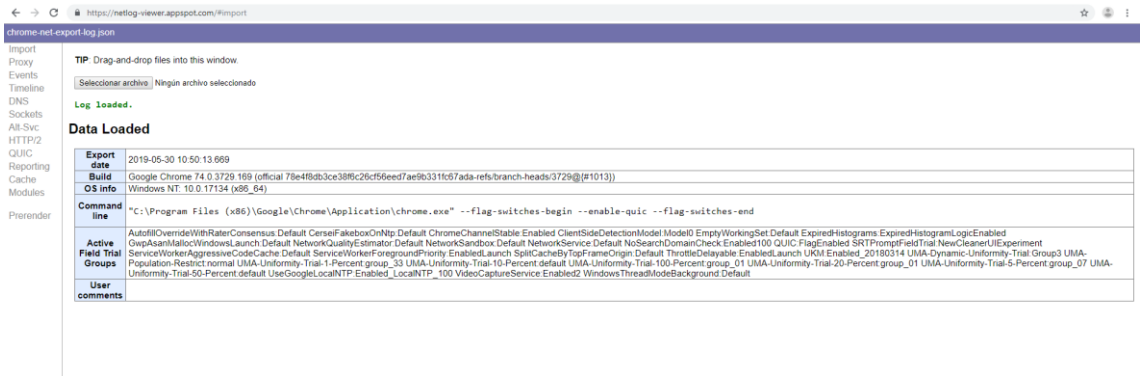


Figura 21: Importación de un registro de red con Chrome.

En primer lugar, accediendo a "Import" se pueden ver algunas de las propiedades del archivo que se ha importado (la fecha en la que se creó, el sistema operativo con el que se realizó…).

El apartado "Proxy" permite ver si se han utilizado proxys durante la navegación web que se ha guardado en el fichero, y en caso afirmativo, muestra cuáles son.

Chrome utiliza un sistema de eventos. Estos eventos son notificaciones que se crean para informar de que ha ocurrido algo interesante durante la navegación, es decir, son como un sistema de alarmas. En esta herramienta, en el campo "Events" se pueden apreciar estos eventos. En la Figura 22 se pueden ver algunos ejemplos de eventos, como, por ejemplo: la petición de una URL, el almacenamiento de datos en la memoria caché, la

dirección de un socket empleado para una comunicación vía UDP, una sesión QUIC (en Chrome las menciones a QUIC se refieren a la versión de Google de QUIC)…

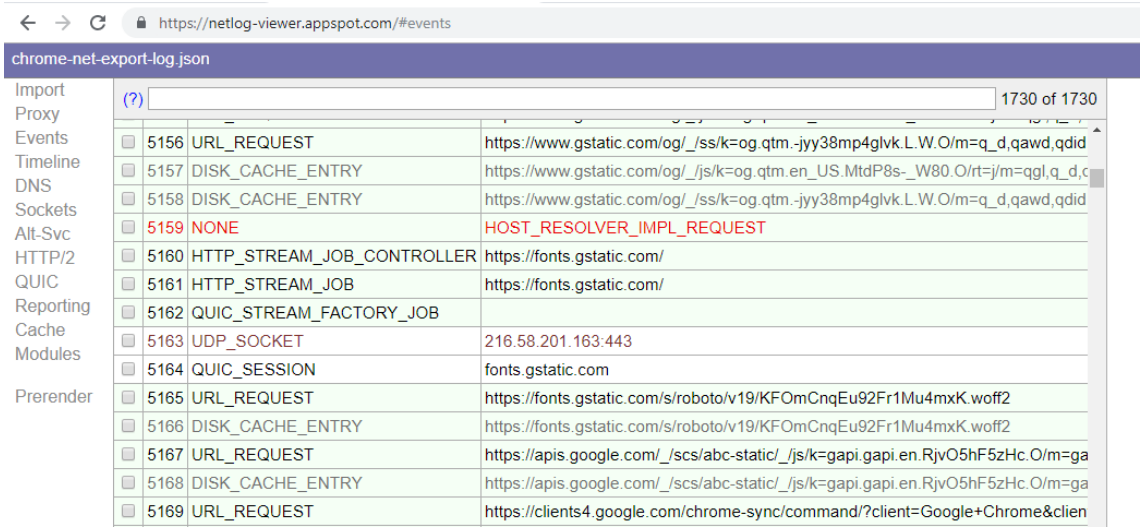
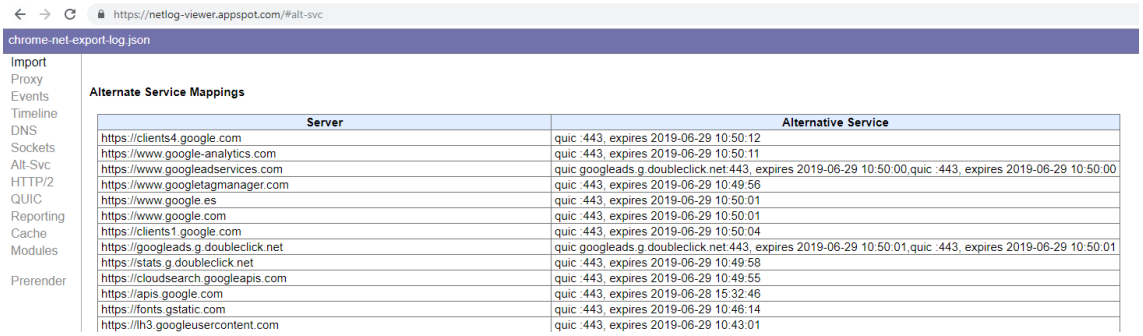


Figura 22: Eventos en Chrome.

En el campo “Timeline” se puede ver la cronología de los eventos y de la información intercambiada entre el cliente y servidor.

En las secciones “DNS” y “Socket” se pueden ver las distintas DNS y Sockets respectivamente que se han utilizado durante la comunicación que se capturó en el fichero que se está analizando.

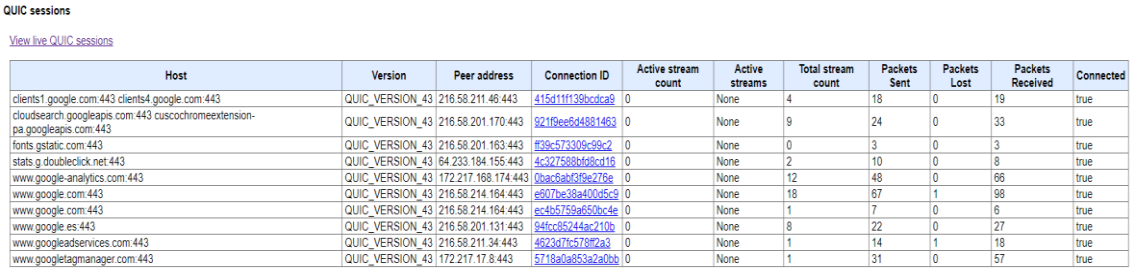
En el capítulo 3.6, se vio cómo es posible la compatibilidad entre HTTP/TLS/TCP y QUIC, la primera petición se hace con HTTP/TLS/TCP y el receptor responde con una cabecera HTTP alt-srv indicando que soporta QUIC. En el apartado “Alt-Svc” se guarda la información de que el servidor es compatible con GQUIC porque se ha recibido una cabecera alt-srv de ese servidor anunciando su compatibilidad. Esto permite al navegador saber que las comunicaciones con ese servidor las puede llevar a cabo utilizando GQUIC. Si al intentar establecer conexión mediante GQUIC, con un servidor que anunció GQUIC en su alt-srv, no se puede, entonces la entrada seguirá existiendo pero con el mensaje “Broken untill…”.



Server	Alternative Service
https://clients4.google.com	quic :443, expires 2019-06-29 10:50:12
https://www.google-analytics.com	quic :443, expires 2019-06-29 10:50:11
https://www.googleadservices.com	quic googleads.g.doubleclick.net:443, expires 2019-06-29 10:50:00,quic :443, expires 2019-06-29 10:50:00
https://www.googletagmanager.com	quic :443, expires 2019-06-29 10:49:56
https://www.google.es	quic :443, expires 2019-06-29 10:50:01
https://www.google.com	quic :443, expires 2019-06-29 10:50:01
https://clients1.google.com	quic :443, expires 2019-06-29 10:50:04
https://googleads.g.doubleclick.net	quic googleads.g.doubleclick.net:443, expires 2019-06-29 10:50:01,quic :443, expires 2019-06-29 10:50:01
https://stats.g.doubleclick.net	quic :443, expires 2019-06-29 10:49:58
https://cloudsearch.googleapis.com	quic :443, expires 2019-06-29 10:49:55
https://apis.google.com	quic :443, expires 2019-06-28 15:32:46
https://fonts.gstatic.com	quic :443, expires 2019-06-29 10:46:14
https://lh3.googleusercontent.com	quic :443, expires 2019-06-29 10:43:01

Figura 23: Servicios alternativos de Chrome.

En los campos “HTTP/2” y “QUIC” se pueden ver las diferentes sesiones, y algunas de sus propiedades, que se han llevado a cabo mediante el protocolo HTTP/2 y QUIC respectivamente. En la Figura 24 se ve un ejemplo de sesiones QUIC capturadas. Se puede ver también un hipervínculo con el texto “View live QUIC sessions”. Al presionar este hipervínculo, se accede al listado de eventos mencionado anteriormente, filtrando únicamente las sesiones QUIC que permanecen activas. También es posible acceder a esta información desde la sección “Events” y escribiendo esta línea “type:QUIC_SESSION is:active” que aplica el mismo filtro. En este listado de eventos, se pueden seleccionar las sesiones QUIC y se muestra una descripción de los mensajes QUIC intercambiados durante la comunicación.



Host	Version	Peer address	Connection ID	Active stream count	Active streams	Total stream count	Packets Sent	Packets Lost	Packets Received	Connected
clients1.google.com:443 clients4.google.com:443	QUIC_VERSION_43	216.58.211.46:443	415d11f139bcdca9	0	None	4	18	0	19	true
cloudsearch.googleapis.com:443 cuscochromeextension-pa.googleapis.com:443	QUIC_VERSION_43	216.58.201.170:443	921f9ee6d4881463	0	None	9	24	0	33	true
fonts.gstatic.com:443	QUIC_VERSION_43	216.58.201.163:443	ff39c573309c99c2	0	None	0	3	0	3	true
stats.g.doubleclick.net:443	QUIC_VERSION_43	64.233.184.155:443	4c327588bfd8cd16	0	None	2	10	0	8	true
www.google-analytics.com:443	QUIC_VERSION_43	172.217.168.174:443	0bac8abf3f9e276e	0	None	12	48	0	66	true
www.google.com:443	QUIC_VERSION_43	216.58.214.164:443	e607be38a400d5c9	0	None	18	67	1	98	true
www.google.com:443	QUIC_VERSION_43	216.58.214.164:443	ec4b5759a650bc4e	0	None	1	7	0	6	true
www.google.es:443	QUIC_VERSION_43	216.58.201.131:443	94fcc85244ac210b	0	None	8	22	0	27	true
www.googleadservices.com:443	QUIC_VERSION_43	216.58.211.34:443	4623d7fc578ff2a3	0	None	1	14	1	18	true
www.googletagmanager.com:443	QUIC_VERSION_43	172.217.17.8:443	5718a0a853a2a0bb	0	None	1	31	0	57	true

Figura 24: Sesiones QUIC activas capturadas con Chrome.

Por último, en los siguientes campos se pueden ver los reportes que se han realizado mientras se capturaba el tráfico, algunas estadísticas de la memoria caché utilizada y las extensiones presentes en el navegador durante la creación del registro de red.

5.2.2 EXTENSIONES EN GOOGLE CHROME

Google Chrome tiene un sistema de extensiones que se pueden instalar en el navegador con la finalidad de realizar tareas concretas. Una de estas extensiones es la llamada "HTTP Indicator". Esta extensión permite ver rápidamente si una página web está utilizando QUIC o no para mostrar su contenido. Para añadirla a nuestro navegador, basta con acceder a https://chrome.google.com/webstore/detail/http-indicator/hgcomhbcacfkpffiphlmnlhpppcjgmbl y presionar el botón "Añadir a Chrome".

Al añadir esta extensión al navegador, aparecerá un icono con forma de rayo en la barra de direcciones web. Este rayo cambiará de color según el protocolo utilizado por la página. El color rojo indica que la página está utilizando QUIC, el azul indica que se está utilizando HTTP/2 y el color gris indica que la web no soporta ninguno de estos protocolos. En la Figura 25 se pueden ver los distintos colores de rayos tras acceder a: Google.com (rayo rojo indicando que se está usando QUIC), a la izquierda, y a marca.com (rayo azul indicando que se está usando HTTP/2), a la derecha.

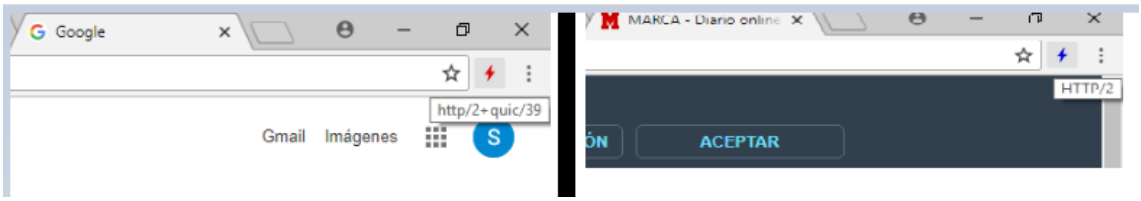


Figura 25: Extensión Google Chrome.

Como se puede ver en la Figura 25, además de indicar que se ha utilizado el protocolo QUIC, también se indica su versión (en este caso la 39).

Además de esta extensión, hay otra muy similar llamada "HTTP/2 and SPDY indicador", que de la misma forma que la extensión anterior, indica con un dibujo de un rayo que toma distintos colores si la comunicación con la página se está llevando a cabo mediante HTTP/2, QUIC o SPDY.

5.3 GQUIC EN SOFTWARE DE SERVIDORES WEB

Los software de servidores web son programas informáticos que reciben peticiones de clientes y envían respuestas con contenido web. Algunos ejemplos son los servidores de código abierto "Apache", "IIS" o "NGINX". Sin embargo, estos servidores aún no

cuentan con mecanismos para manejar tráfico QUIC. Dos servidores web que son compatibles con QUIC actualmente son “Caddy” [24] y “LiteSpeed” [25], aunque no son tan conocidos como los anteriores.

Para probar la conexión con un software de servidor web se ha desplegado un servidor Apache, y se ha procedido a mirar el tráfico. Para ello se ha descargado el código fuente, ejecutables y archivos necesarios. Ha sido necesario cambiar la raíz del servidor en el código, por la ubicación en la que se encuentra el servidor en mi equipo. También ha sido necesario modificar el nombre del servidor (se ha llamado localhost) y como puerto de escucha se ha elegido el 443. Una vez realizadas estas modificaciones en la configuración, se ha ejecutado el servidor y se ha accedido mediante Chrome a “localhost:443” como puede verse en la Figura 26 (se han ocultado los marcadores del navegador).

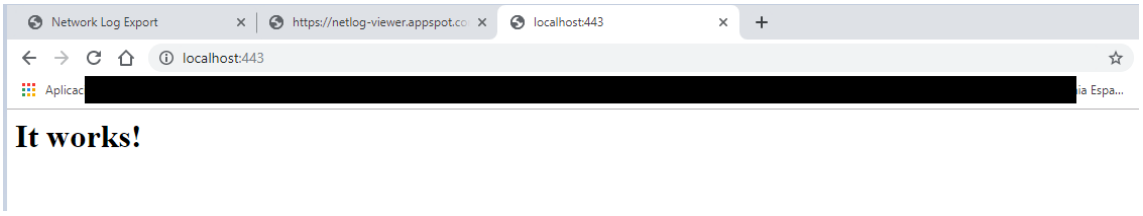


Figura 26: Conexión con servidor Apache.

Al llevar a cabo la conexión con este servidor, se ha capturado el tráfico como se indica en el punto 5.2.1. En la Figura 27 puede verse cómo las únicas sesiones QUIC provienen de servidores de Google y no del servidor apache.

QUIC sessions

View live QUIC sessions

Host	Version	Peer address	Connection ID	Active stream count	Active streams
cloudsearch.googleapis.com:443 cuscochromeextension-pa.googleapis.com:443	QUIC_VERSION_43	172.217.17.10:443	6475a91d337ea307	0	None
www.google.com:443	QUIC_VERSION_43	172.217.16.228:443	7cb1c73b917710df	0	None

Figura 27: Sesiones QUIC con servidor Apache.

5.4 COMPATIBILIDAD DE GQUIC CON SITIOS WEB

En esta sección se estudia la compatibilidad de la versión de QUIC de Google con diferentes servidores web. Para ello se han llevado a cabo dos pruebas diferentes.

5.4.1 METODOLOGÍA

En la primera prueba, enfocada a comprobar la compatibilidad de GQUIC con los principales servidores web, se ha utilizado la herramienta Alexa de Amazon para ver el top 15 de páginas web más visitadas en todo el mundo. A continuación, se ha establecido conexión con cada una de ellas utilizando Google Chrome, y se ha utilizado el analizador de protocolos Wireshark para capturar los paquetes intercambiados y comprobar cuáles generan tráfico GQUIC.

Es importante destacar que durante el análisis del tráfico en algunas páginas que no soportan GQUIC se ha obtenido tráfico GQUIC. Por tanto, ha sido necesario analizar las IPs de las que provenía este tráfico GQUIC (utilizando el servicio whois explicado en el capítulo 5.5.2). Se han producido casos en los que la página no soportaba GQUIC y no pertenecía a Google, pero había tráfico GQUIC que venía de servidores de Google. Esto quiere decir que, aunque una página lleve a cabo el intercambio de contenido con un cliente mediante HTTP/2 y TCP, si esa página tiene anuncios u otros contenidos que están alojados en servidores que utilizan GQUIC, la descarga de esos contenidos se producirá mediante GQUIC. Por tanto, para la prueba, no basta con ver si se genera tráfico GQUIC, sino que también hay que analizar el origen para ver si se corresponde con la página analizada.

La segunda prueba se ha enfocado en estudiar la compatibilidad de GQUIC con otras páginas web más pequeñas. Debido a que hasta ahora sólo conocíamos páginas web de Google que soporten GQUIC (Google, Gmail, Youtube…), teníamos interés en encontrar otros sitios web que pudieran usar esta misma versión del protocolo. Encontramos en [26] estadísticas del uso de QUIC, además de una lista con las páginas web que soportan QUIC más utilizadas, otra lista con páginas aleatorias que utilizan QUIC y una última lista con páginas que recientemente han incorporado el uso de QUIC.

Para probar si estas páginas soportan GQUIC, se ha creado y visualizado un fichero con los registros de red de Chrome tras acceder a todas ellas. En este registro, se han comprobado dos cosas: si las páginas ofrecen QUIC como servicio alternativo para la comunicación y si la comunicación realmente se lleva a cabo utilizando GQUIC.

Para comprobar si las páginas ofrecen QUIC como servicio alternativo basta con mirar el apartado Alt-Srv del registro de red. Sin embargo, en muchas de las páginas que sí ofrecen este servicio alternativo se indica “Broken until…”, esto quiere decir que ese servidor se

anuncia como compatible con QUIC mediante una cabecera alt-srv como se ha explicado antes, pero luego no se puede establecer una sesión GQUIC con ellas. Esto puede ser por varios motivos, porque el servidor realmente no soporta GQUIC aunque lo anuncie con un alt-srv, por incompatibilidad en la versión del protocolo, o porque se filtre el tráfico UDP en algún punto de la comunicación.

Cualquiera de estas dos pruebas podría usarse tanto como para probar el uso de GQUIC con los sitios web más visitados como con las páginas de menor tamaño. Sin embargo, se ha decidido realizar una prueba diferente para sitios grandes y pequeños para dar una visión más amplia.

5.4.2 RESULTADOS

La primera prueba trataba de comprobar la compatibilidad de GQUIC con las 15 páginas web más visitadas en el mundo. En la Tabla 4 se recogen estas páginas por orden y si soportan o no GQUIC:

Página web	¿soporta GQUIC?
Google.com	Sí
Youtube.com	Sí
Facebook.com	No
Baidu.com	No
Wikipedia.org	No
Qq.com	No
Tmall.com	No
Taobao.com	No
Yahoo.com	No
Amazon.com	No
Twitter.com	No
Sohu.com	No
Instagram.com	No
Live.com	No
Jd.com	No

Tabla 4: GQUIC en las 15 páginas web más utilizadas.

Como se puede ver, GQUIC sólo está siendo utilizado actualmente por dos de las páginas web más visitadas (ambas de Google). Sin embargo, estas dos páginas son las más utilizadas. Por tanto, aunque GQUIC no está siendo utilizado actualmente en la mayoría del tráfico de Internet, sí representa un porcentaje elevado del tráfico web mundial.

En la segunda prueba se ha evaluado la compatibilidad de GQUIC con algunas páginas más pequeñas que se anuncian en [26] como compatibles con el protocolo.

En la Tabla 5, se recogen los resultados de esta prueba:

Página web	**¿anuncia QUIC cómo alt-srv?**	**¿realmente se lleva a cabo una comunicación mediante GQUIC?**
chaneldelisser.com	Sí	Sí
testerwork.com	Sí	No
accelerate. education	Sí	Sí
chetor.com	Sí	No
softfamous.com	Sí	No
Kamupersoneli.net	Sí	Sí
Arga-mag.com	Sí	No
Cepmarket.com.tr	Sí	Sí
Annuaire-telechargement.com	Sí	Sí
Cloverseniorsupport.com	Sí	Sí
Interstellarorchard.com	Sí	Sí
Telluriumq.com	Sí	Sí
Hostplay.com	Sí	No

Tabla 5: GQUIC en páginas pequeñas.

Como puede verse, todas las páginas que se han probado sí que anuncian QUIC como protocolo alternativo, sin embargo, sólo ocho de trece han sido capaces realmente de llevar a cabo la comunicación utilizando GQUIC.

En la Figura 28 y la Figura 29 se pueden ver subrayadas en amarillo las páginas que se han comprobado. Como se observa, todas ellas anuncian QUIC como servicio alternativo, pero en varias de ellas aparece el mensaje “Broken untill…” y no han utilizado GQUIC para la comunicación.

Alternate Service Mappings

Server	Alternative Service
https://www.google-analytics.com	quic :443, expires 2019-07-02 11:38:12
https://softfamous.com	quic :443, expires 2019-06-03 11:38:12 (broken until 2019-06-02 11:41:3)
https://fonts.gstatic.com	quic :443, expires 2019-07-02 11:38:04
https://fonts.googleapis.com	quic :443, expires 2019-07-02 11:35:59
https://www.googletagmanager.com	quic :443, expires 2019-07-02 11:35:59
https://ad.doubleclick.net	quic :443, expires 2019-06-30 13:11:05
https://pagead2.googlesyndication.com	quic googleads.g.doubleclick.net:443, expires 2019-06-30 14:11:06,quic :443, expires 2019-06-30 14:11:06
https://www.youtube.com	quic :443, expires 2019-07-02 11:38:09
https://www.chetor.com	quic :443, expires 2019-07-02 11:38:09 (broken until 2019-06-02 11:40:55)
https://accelerate.education	quic :443, expires 2019-06-03 11:38:04
https://testerwork.com	quic :443, expires 2019-06-03 11:37:48 (broken until 2019-06-04 6:15:20)
https://chaneldelisser.com	quic :443, expires 2019-06-03 11:37:38

Figura 28: Otras páginas que soportan QUIC.

Alternate Service Mappings

Server	Alternative Service
https://googleads.g.doubleclick.net	quic googleads.g.doubleclick.net:443, expires 2019-07-14 12:59:28,quic :443, expires 2019-07-14 12:59:28
https://www.kamupersoneli.net	quic :443, expires 2019-07-14 12:59:21
https://adservice.google.com	quic googleads.g.doubleclick.net:443, expires 2019-07-14 12:59:18,quic :443, expires 2019-07-14 12:59:18
https://adservice.google.es	quic googleads.g.doubleclick.net:443, expires 2019-07-14 12:59:18,quic :443, expires 2019-07-14 12:59:18
https://www.googletagmanager.com	quic :443, expires 2019-07-14 12:59:14
https://arga-mag.com	quic :443, expires 2019-07-14 12:59:13 (broken until 2019-06-14 13:8:52)
https://cepmarket.com.tr	quic :443, expires 2019-07-14 12:58:46
https://www.youtube.com	quic :443, expires 2019-07-14 12:58:35
https://accounts.google.com	quic :443, expires 2019-07-14 12:58:26
https://wwwwv.annuaire-telechargement.com	quic :443, expires 2019-07-14 12:58:09
https://www.cloverseniorsupport.com	quic :443, expires 2019-07-14 12:57:32
https://static.doubleclick.net	quic :443, expires 2019-07-14 12:57:10
https://interstellarorchard.com	quic :443, expires 2019-06-15 12:56:09
https://telluriumq.com	quic :443, expires 2019-06-15 12:55:07
https://maps.googleapis.com	quic :443, expires 2019-07-14 12:55:06
https://translate.googleapis.com	quic :443, expires 2019-07-14 12:54:59
https://translate.google.com	quic :443, expires 2019-07-14 12:54:43
https://cm.g.doubleclick.net	quic googleads.g.doubleclick.net:443, expires 2019-07-14 12:53:24,quic :443, expires 2019-07-14 12:53:24
https://clients2.google.com	quic :443, expires 2019-07-14 12:51:36
https://www.hostplay.com	quic :443, expires 2019-07-14 13:15:56 (broken until 2019-06-14 13:26:0)

Figura 29: Más páginas pequeñas que soportan QUIC.

5.5 COMPATIBILIDAD DE GQUIC CON REDES DE DISTRIBUCIÓN DE CONTENIDOS

Las redes de distribución de contenidos son sistemas de servidores que se encuentran repartidos geográficamente para servir contenidos de sitios web. Es decir, en lugar de tener toda la información de Google, por ejemplo, en un único servidor en Estados Unidos, esta información se duplica y se almacena en un gran número de servidores alojados en diferentes regiones y países a lo largo del globo terrestre. Luego, las peticiones al sitio web se redirigen (empleando diferentes técnicas, por ejemplo, las respuestas DNS) al servidor más cercano geográficamente al cliente que ha hecho la petición. De esta forma se evita la sobrecarga de los servidores. Además, al haber servidores con la misma información más cerca geográficamente del cliente, se reduce el tiempo de respuesta. Google tiene su propia red de distribución para sus propios

contenidos, pero también existen empresas que ofrecen el servicio a sitios web que quieran disfrutar de las ventajas de las redes de distribución de contenido sin tener que desplegar una infraestructura propia [27].

Un ejemplo de red de distribución de contenido es la empresa "Akamai". Esta compañía comenzó a estudiar el uso del protocolo QUIC en julio de 2016 y desde entonces ha ido evolucionando debido a que la compañía ha visto las ventajas que este protocolo aporta a las comunicaciones entre sus servidores y los clientes. Además, Akamai está participando en el desarrollo de la versión de QUIC del IETF, e incorporará dicha versión a sus servidores cuando se estandarice [28].

Otra empresa dedicada a los servidores de distribución de contenidos, es "Cloudflare", la cual incorporará QUIC a sus servidores cuando se estandarice la versión del IETF [29]. Se hablará de esta compañía en el capítulo 5.7.

5.5.1 METODOLOGÍA

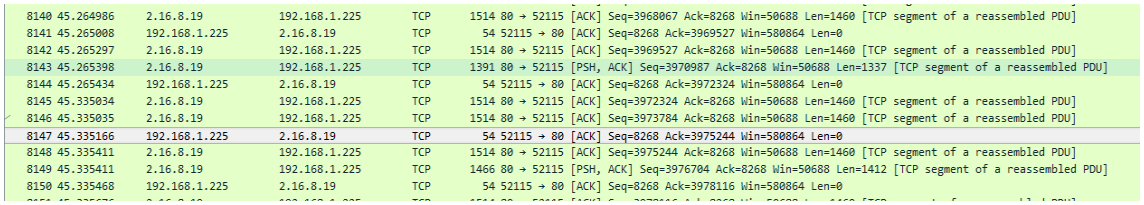
Queremos ver si el interés mostrado por Akamai en el protocolo se ha traducido en la implementación del mismo en sus servidores. Para probar la compatibilidad de Akamai con GQUIC se ha probado a establecer conexión con cinco dominios de Akamai desde Google Chrome (akaquic.com, akamai.com, Apple.com, paypal.com y bbva.es). Mediante Wireshark se ha comprobado el protocolo utilizado en estas conexiones. Es importante destacar que esta prueba no demuestra que no se soporte GQUIC en ningún servidor de Akamai, pero en todo caso su uso no está extendido.

Como ocurría en el capítulo 5.4, al realizar esta prueba se ha producido tráfico GQUIC en Wireshark proveniente de diferentes direcciones IP de Google. Se ha creado un registro de red y se ha analizado para ver si proviene de anuncios de Google o de páginas de Akamai.

5.5.2 RESULTADOS

En la Figura 30, puede verse el intercambio de paquetes con el servidor de "akaquic.com" cuya dirección IP es 2.16.8.0. Este intercambio se lleva a cabo mediante TCP y no mediante GQUIC.

Utilizando el servicio whois (en la página web https://bandaancha.eu/whois) se puede introducir esta IP y verificar que pertenece a Akamai. Figura 31.



8140	45.264986	2.16.8.19	192.168.1.225	TCP	1514 80 → 52115 [ACK] Seq=3968067 Ack=8268 Win=50688 Len=1460 [TCP segment of a reassembled PDU]
8141	45.265008	192.168.1.225	2.16.8.19	TCP	54 52115 → 80 [ACK] Seq=8268 Ack=3969527 Win=580864 Len=0
8142	45.265297	2.16.8.19	192.168.1.225	TCP	1514 80 → 52115 [ACK] Seq=3969527 Ack=8268 Win=50688 Len=1460 [TCP segment of a reassembled PDU]
8143	45.265398	2.16.8.19	192.168.1.225	TCP	1391 80 → 52115 [PSH, ACK] Seq=3970987 Ack=8268 Win=50688 Len=1337 [TCP segment of a reassembled PDU]
8144	45.265434	192.168.1.225	2.16.8.19	TCP	54 52115 → 80 [ACK] Seq=8268 Ack=3972324 Win=580864 Len=0
8145	45.335034	2.16.8.19	192.168.1.225	TCP	1514 80 → 52115 [ACK] Seq=3972324 Ack=8268 Win=50688 Len=1460 [TCP segment of a reassembled PDU]
8146	45.335035	2.16.8.19	192.168.1.225	TCP	1514 80 → 52115 [ACK] Seq=3973784 Ack=8268 Win=50688 Len=1460 [TCP segment of a reassembled PDU]
8147	45.335166	192.168.1.225	2.16.8.19	TCP	54 52115 → 80 [ACK] Seq=8268 Ack=3975244 Win=580864 Len=0
8148	45.335411	2.16.8.19	192.168.1.225	TCP	1514 80 → 52115 [ACK] Seq=3975244 Ack=8268 Win=50688 Len=1460 [TCP segment of a reassembled PDU]
8149	45.335411	2.16.8.19	192.168.1.225	TCP	1466 80 → 52115 [PSH, ACK] Seq=3976704 Ack=8268 Win=50688 Len=1412 [TCP segment of a reassembled PDU]
8150	45.335468	192.168.1.225	2.16.8.19	TCP	54 52115 → 80 [ACK] Seq=8268 Ack=3978116 Win=580864 Len=0

Figura 30: Conexión con Akamai.

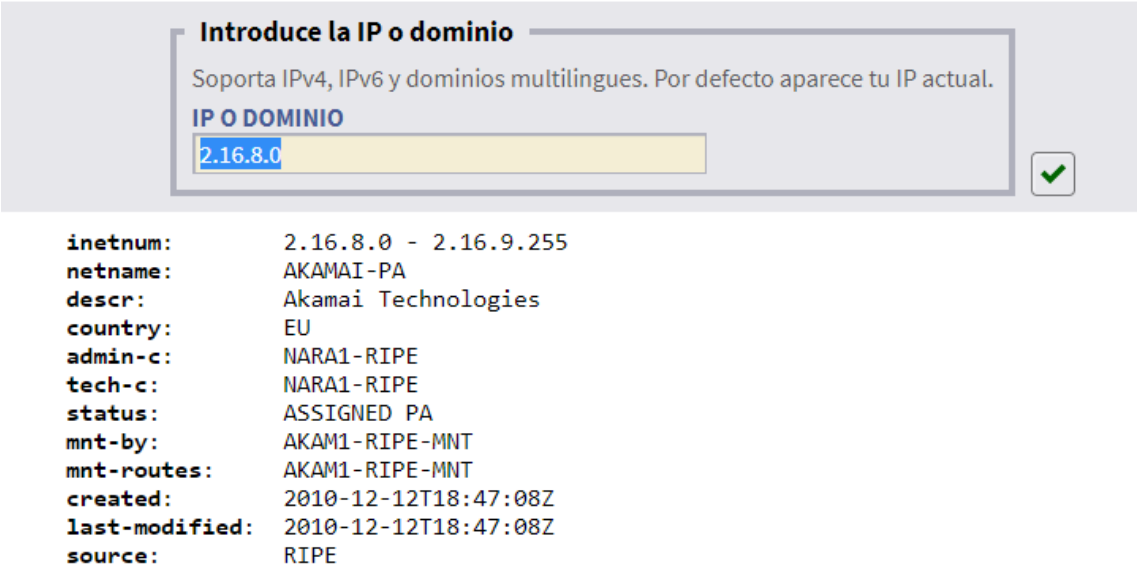
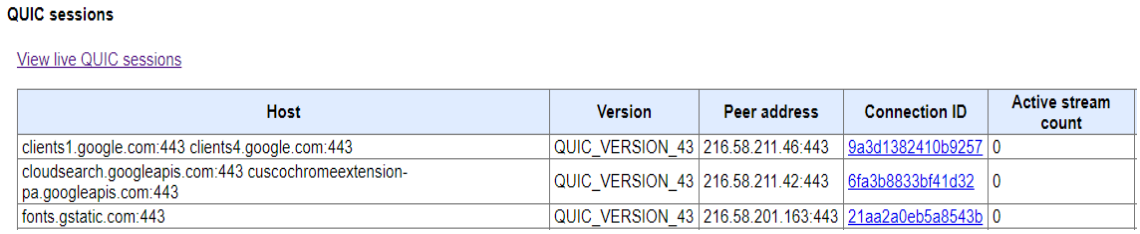


Figura 31: IP Akamai.

Como puede verse en la Figura 32, aunque la conexión con el servidor de Akamai se lleve a cabo mediante HTTP/2 y TCP, hay sesiones QUIC activas. Todas estas sesiones provienen de anuncios y servidores de Google.



QUIC sessions

View live QUIC sessions

Host	Version	Peer address	Connection ID	Active stream count
clients1.google.com:443 clients4.google.com:443	QUIC_VERSION_43	216.58.211.46:443	9a3d1382410b9257	0
cloudsearch.googleapis.com:443 cuscochromeextension-pa.googleapis.com:443	QUIC_VERSION_43	216.58.211.42:443	6fa3b8833bf41d32	0
fonts.gstatic.com:443	QUIC_VERSION_43	216.58.201.163:443	21aa2a0eb5a8543b	0

Figura 32: Sesiones QUIC al conectarse a Akamai.

Los resultados obtenidos son los mismos para los otros dominios comprobados (akamai.com, Apple.com, paypal.com y bbva.es). Con esto no se ha demostrado que

Akamai no haya implementado el uso de GQUIC en ninguno de sus servidores, pero en caso de haberlo hecho, su uso no se ha extendido.

5.6 ANÁLISIS TÉCNICO DEL TRÁFICO GQUIC EN INTERNET

Para terminar con el análisis de GQUIC, en esta sección se estudia el comportamiento técnico del protocolo en Internet para comprobar si coincide con la teoría explicada en el capítulo 3.

5.6.1 HANDSHAKES

Se va a evaluar el intercambio de paquetes a la hora de establecer una conexión entre un cliente y un servidor utilizando el protocolo QUIC desarrollado por Google. Como cliente se utilizará un ordenador con el navegador Google Chrome, y en el lado del servidor, estará el servidor de Gmail.

En primer lugar, se va a analizar el intercambio de paquetes requerido para establecer una conexión cuando no ha habido ninguna conexión previa reciente, es decir, el handshake explicado en el capítulo 3.7 de 1RTT.

No.	Time	Source	Destination	Protocol	Length	Info
2576	21.598426	10.118.23.222	172.217.17.3	GQUIC	1392	Client Hello, PKN: 1, CID: 11634847808494004003
2629	21.633727	172.217.17.3	10.118.23.222	GQUIC	1392	Rejection, PKN: 1, CID: 11634847808494004003
2637	21.636387	10.118.23.222	172.217.17.3	GQUIC	1392	Client Hello, PKN: 2, CID: 11634847808494004003
2638	21.636706	10.118.23.222	172.217.17.3	GQUIC	870	Payload (Encrypted), PKN: 3, CID: 11634847808494004003
2646	21.662069	172.217.17.3	10.118.23.222	GQUIC	1392	Payload (Encrypted), PKN: 2
2647	21.662071	172.217.17.3	10.118.23.222	GQUIC	73	Payload (Encrypted), PKN: 3
2648	21.663464	10.118.23.222	172.217.17.3	GQUIC	83	Payload (Encrypted), PKN: 4, CID: 11634847808494004003
2649	21.663619	10.118.23.222	172.217.17.3	GQUIC	70	Payload (Encrypted), PKN: 5, CID: 11634847808494004003

Figura 33: Captura de tráfico GQUIC handshake sin conexión previa reciente.

Como se puede ver en la Figura 33, en primer lugar, el cliente envía un mensaje Client Hello incompleto. Por tanto, el servidor responde con un mensaje “Rejection”. Ahora el cliente tiene la información que necesitaba para la conexión y envía de nuevo un Client Hello, esta vez completo, e inmediatamente después envía la petición del recurso. Por tanto, se tarda 1 RTT en enviar datos.

Ahora se realizará la misma prueba, pero con un servidor con el cual se ha llevado a cabo una conexión reciente.

No.	Time	Source	Destination	Protocol	Length	Info
1013	10.291523	10.118.23.222	216.58.201.132	GQUIC	1392	Client Hello, PKN: 1, CID: 15318171583759180986
1014	10.291679	10.118.23.222	216.58.201.132	GQUIC	494	Payload (Encrypted), PKN: 2, CID: 15318171583759180986
1015	10.317784	216.58.201.132	10.118.23.222	GQUIC	1392	Payload (Encrypted), PKN: 1
1016	10.318792	10.118.23.222	216.58.201.132	GQUIC	83	Payload (Encrypted), PKN: 3, CID: 15318171583759180986
1017	10.318902	216.58.201.132	10.118.23.222	GQUIC	73	Payload (Encrypted), PKN: 2
1018	10.318991	10.118.23.222	216.58.201.132	GQUIC	70	Payload (Encrypted), PKN: 4, CID: 15318171583759180986
1019	10.321893	216.58.201.132	10.118.23.222	GQUIC	62	Payload (Encrypted), PKN: 3
1026	10.345617	10.118.23.222	216.58.201.132	GQUIC	70	Payload (Encrypted), PKN: 5, CID: 15318171583759180986
1027	10.358905	216.58.201.132	10.118.23.222	GQUIC	1181	Payload (Encrypted), PKN: 4
1028	10.359590	216.58.201.132	10.118.23.222	GQUIC	60	Payload (Encrypted), PKN: 5
1029	10.360341	10.118.23.222	216.58.201.132	GQUIC	70	Payload (Encrypted), PKN: 6, CID: 15318171583759180986

Figura 34: Captura de tráfico GQUIC handshake con conexión previa reciente.

Como se puede ver en la Figura 34, el tiempo necesario es 0 RTT, ya que se envía el Client Hello del handshake e inmediatamente después se envía la solicitud del recurso y el servidor responde a dicha solicitud. Como se explicó en el punto 3.7, esto es posible debido a que el cliente guarda los datos obtenidos del servidor en una conexión anterior en su memoria caché.

5.6.2 SERVIDORES DESCONOCIDOS

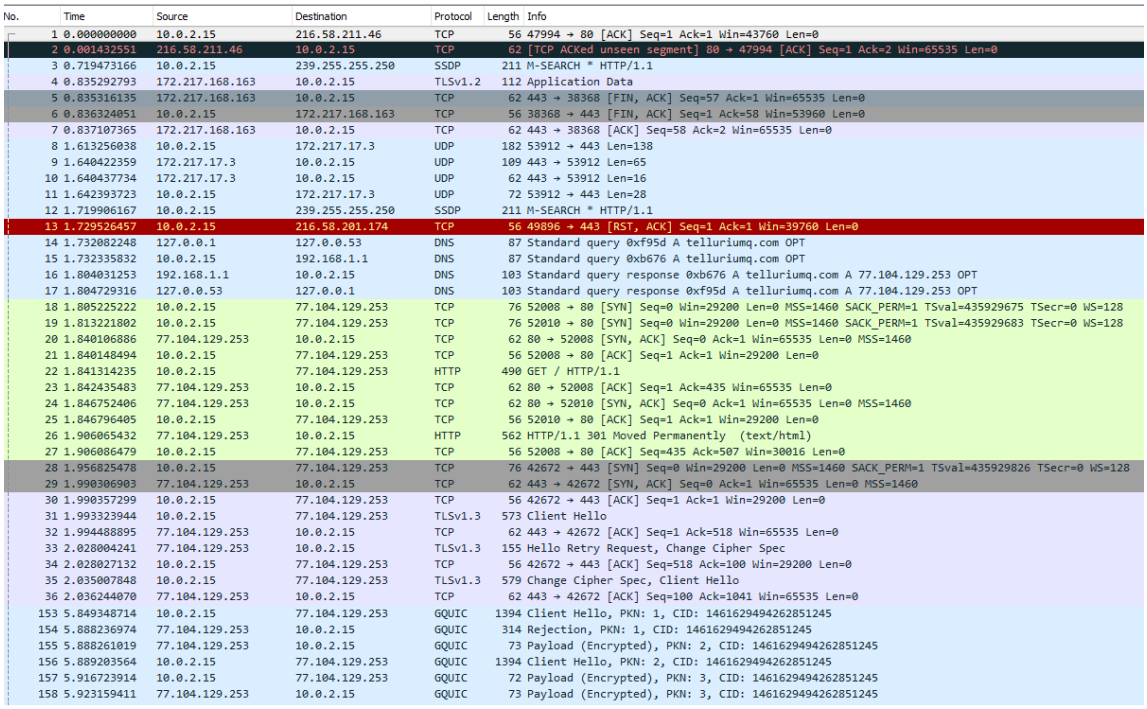
En este apartado se va a comprobar cómo es el establecimiento de conexión entre un cliente y un servidor que no han tenido ninguna conexión previa nunca. Por tanto, al realizar la petición el cliente, no sabe si el servidor soporta o no GQUIC.

Como se explica en el punto 3.6, el cliente envía una primera solicitud mediante TLS, TCP y HTTP, y el servidor, en su respuesta, le indica que puede utilizar QUIC mediante el campo alt-srv.

En lugar de llevar a cabo una conexión con un servidor con el que no se había conectado nunca antes, también se podrían borrar todos los datos que el navegador tiene guardados de una web. Esto puede hacerse accediendo al historial de Chrome, seleccionando el botón “Borrar datos de navegación” y, para asegurarnos de que no queda ninguna conexión con ese servidor ni ningún otro, seleccionamos intervalo de tiempo: “desde el origen de los tiempos” y en la pestaña de configuración avanzada seleccionamos todas las opciones y pulsamos “borrar datos”.

Para probarlo con un servidor completamente desconocido para nuestro navegador, se ha establecido conexión con telluriumq.com utilizando Google Chrome. Como se puede ver en la Figura 35, el cliente lleva a cabo una petición HTTP sobre TCP y TLS 1.3 y al recibir datos del servidor, como se explicó en el capítulo 3.6, gracias a la cabecera alt-srv

sabe que el servidor soporta GQUIC. Por tanto, el siguiente mensaje es un Client Hello de GQUIC.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	216.58.211.46	TCP	56	47994 → 80 [ACK] Seq=1 Ack=1 Win=43760 Len=0
2	0.001432551	216.58.211.46	10.0.2.15	TCP	62	[TCP ACKed unseen segment] 80 → 47994 [ACK] Seq=1 Ack=2 Win=65535 Len=0
3	0.719473166	10.0.2.15	239.255.255.250	SSDP	211	M-SEARCH * HTTP/1.1
4	0.835292793	172.217.168.163	10.0.2.15	TLSv1.2	112	Application Data
5	0.835316135	172.217.168.163	10.0.2.15	TCP	62	443 → 38368 [FIN, ACK] Seq=57 Ack=1 Win=65535 Len=0
6	0.836324051	10.0.2.15	172.217.168.163	TCP	56	38368 → 443 [FIN, ACK] Seq=1 Ack=58 Win=53960 Len=0
7	0.837107365	172.217.168.163	10.0.2.15	TCP	62	443 → 38368 [ACK] Seq=58 Ack=2 Win=65535 Len=0
8	1.613256038	10.0.2.15	172.217.17.3	UDP	182	53912 → 443 Len=138
9	1.640422359	172.217.17.3	10.0.2.15	UDP	109	443 → 53912 Len=65
10	1.640437734	172.217.17.3	10.0.2.15	UDP	62	443 → 53912 Len=16
11	1.642393723	10.0.2.15	172.217.17.3	UDP	72	53912 → 443 Len=28
12	1.719906167	10.0.2.15	239.255.255.250	SSDP	211	M-SEARCH * HTTP/1.1
13	1.729526457	10.0.2.15	216.58.201.174	TCP	56	49896 → 443 [RST, ACK] Seq=1 Ack=1 Win=39760 Len=0
14	1.732082248	127.0.0.1	127.0.0.53	DNS	87	Standard query 0xf95d A telluriumq.com OPT
15	1.732335832	10.0.2.15	192.168.1.1	DNS	87	Standard query 0xb676 A telluriumq.com OPT
16	1.804031253	192.168.1.1	10.0.2.15	DNS	103	Standard query response 0xb676 A telluriumq.com A 77.104.129.253 OPT
17	1.804729316	127.0.0.53	127.0.0.1	DNS	103	Standard query response 0xf95d A telluriumq.com A 77.104.129.253 OPT
18	1.805225222	10.0.2.15	77.104.129.253	TCP	76	52008 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=435929675 TSecr=0 WS=128
19	1.813221802	10.0.2.15	77.104.129.253	TCP	76	52010 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=435929683 TSecr=0 WS=128
20	1.840106886	77.104.129.253	10.0.2.15	TCP	62	80 → 52008 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
21	1.840148494	10.0.2.15	77.104.129.253	TCP	56	52008 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
22	1.841314235	10.0.2.15	77.104.129.253	HTTP	490	GET / HTTP/1.1
23	1.842435483	77.104.129.253	10.0.2.15	TCP	62	80 → 52008 [ACK] Seq=1 Ack=435 Win=65535 Len=0
24	1.846752406	77.104.129.253	10.0.2.15	TCP	62	80 → 52010 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
25	1.846796405	10.0.2.15	77.104.129.253	TCP	56	52010 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
26	1.906065432	77.104.129.253	10.0.2.15	HTTP	562	HTTP/1.1 301 Moved Permanently (text/html)
27	1.906086479	10.0.2.15	77.104.129.253	TCP	56	52008 → 80 [ACK] Seq=435 Ack=507 Win=30016 Len=0
28	1.956825478	10.0.2.15	77.104.129.253	TCP	76	42672 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=435929826 TSecr=0 WS=128
29	1.990306903	77.104.129.253	10.0.2.15	TCP	62	443 → 42672 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
30	1.990357299	10.0.2.15	77.104.129.253	TCP	56	42672 → 443 [ACK] Seq=1 Ack=1 Win=29200 Len=0
31	1.993323944	10.0.2.15	77.104.129.253	TLSv1.3	573	Client Hello
32	1.994488895	77.104.129.253	10.0.2.15	TCP	62	443 → 42672 [ACK] Seq=1 Ack=518 Win=65535 Len=0
33	2.028004241	77.104.129.253	10.0.2.15	TLSv1.3	155	Hello Retry Request, Change Cipher Spec
34	2.028027132	10.0.2.15	77.104.129.253	TCP	56	42672 → 443 [ACK] Seq=518 Ack=100 Win=29200 Len=0
35	2.035007848	10.0.2.15	77.104.129.253	TLSv1.3	579	Change Cipher Spec, Client Hello
36	2.036244070	77.104.129.253	10.0.2.15	TCP	62	443 → 42672 [ACK] Seq=100 Ack=1041 Win=65535 Len=0
153	5.849348714	10.0.2.15	77.104.129.253	GQUIC	1394	Client Hello, PKN: 1, CID: 14616294942628512 45
154	5.888236974	77.104.129.253	10.0.2.15	GQUIC	314	Rejection, PKN: 1, CID: 14616294942628512 45
155	5.888261019	77.104.129.253	10.0.2.15	GQUIC	73	Payload (Encrypted), PKN: 2, CID: 14616294942628512 45
156	5.889203564	10.0.2.15	77.104.129.253	GQUIC	1394	Client Hello, PKN: 2, CID: 14616294942628512 45
157	5.916723914	10.0.2.15	77.104.129.253	GQUIC	72	Payload (Encrypted), PKN: 3, CID: 14616294942628512 45
158	5.923159411	77.104.129.253	10.0.2.15	GQUIC	73	Payload (Encrypted), PKN: 3, CID: 14616294942628512 45

Figura 35: Captura Wireshark web desconocida.

5.6.3 ESTRUCTURA PAQUETES GQUIC EN WIRESHARK

Con ayuda del programa informático Wireshark, se ha capturado tráfico GQUIC y se ha analizado la composición del paquete para ver si coincide con la teoría explicada anteriormente. En la Figura 36 se puede ver la estructura de un paquete GQUIC del tipo "Client Hello".

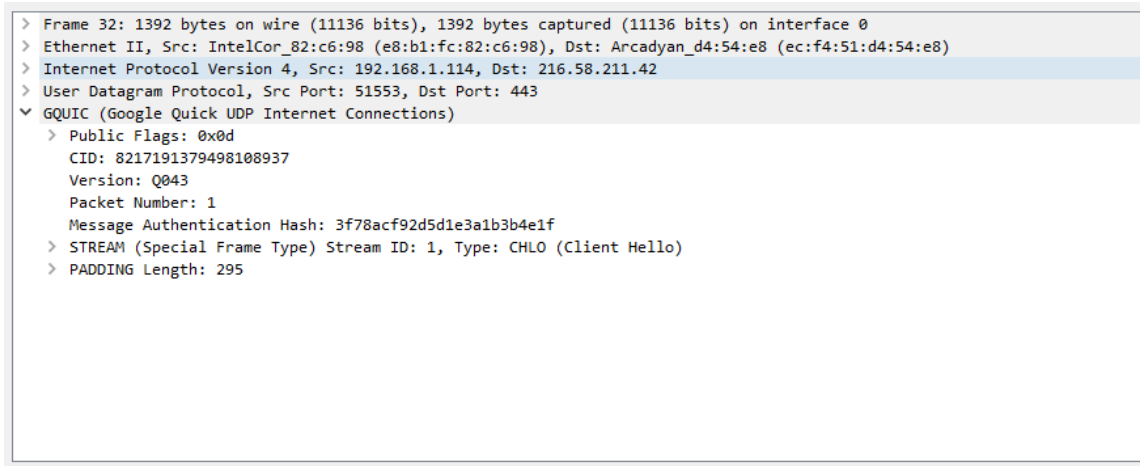


Figura 36: Estructura paquete GQUIC en Wireshark.

Comparando la Figura 36 con la Figura 13, se puede ver que los tres primeros campos coinciden. En primer lugar, está el campo que contiene los "Flags", después el "Connection ID", y a continuación, el "Version". Después de este campo, en la Figura 13 aparece el denominado "Diversification nonce". Sin embargo, este campo es opcional y sólo aparece en los paquetes que el servidor envía al cliente, por tanto, es normal que no esté en este paquete capturado. En cuarto lugar, está el "Packet Number", que también coincide con la estructura de la Figura 13. Por último, se puede ver en este paquete, un nuevo campo llamado "Message Authentication Hash", el cual no aparece en la estructura de la cabecera de la Figura 13. Este campo no aparece en la última versión de la especificación de QUIC en Google, pero en una versión anterior se explica que este campo aparece en paquetes iniciales como "Client Hello", "Server Hello" o "Rejection" debido a que estos paquetes no están encriptados pero incluyen un mensaje Hash.

Después de la cabecera y el "Message Authentication Hash" vienen los distintos streams. Sin embargo, al tratarse del primer paquete (Client Hello) sólo puede ir un stream en dicho paquete (el stream de Crypto).

En la Figura 37 se muestra la estructura de un stream, en la que se pueden ver los campos "Frame type", "stream ID" y "Data length" que se comentaban en el capítulo 3.9.1. Además de los campos mencionados, también aparecen la etiqueta con el tipo de paquete del que se trata (en este caso "Client Hello") y por tratarse de este tipo de paquete, el stream contiene también las características que el cliente comunica al servidor (son los distintos "Tag/value" que se pueden ver en dicha Figura 37.

Como se puede apreciar, entre estos valores están el STK, los algoritmos utilizados para encriptar el mensaje, el número máximo de streams permitidos…

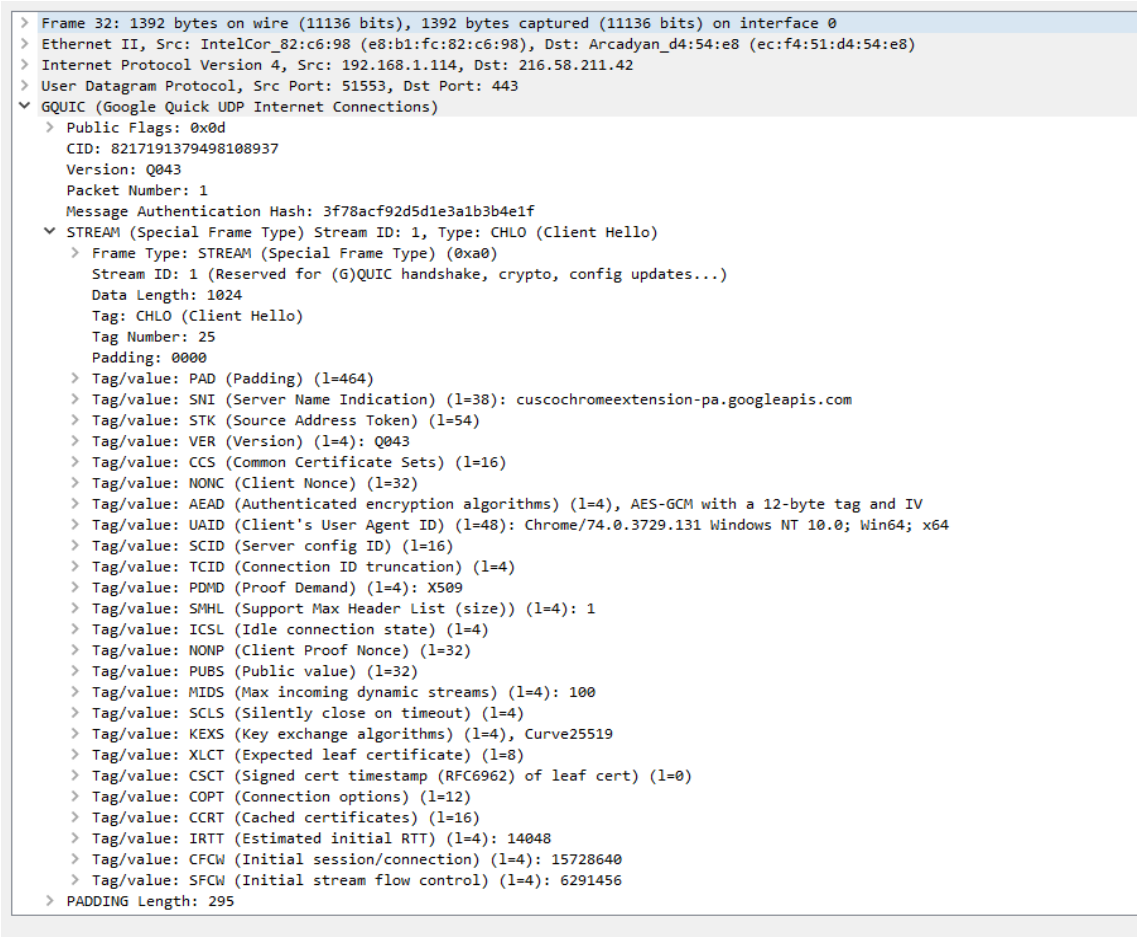


Figura 37: Estructura de un stream en Wireshark.

5.7 ESTUDIO DE LA VERSIÓN DE QUIC DEL IETF

Como se ha mencionado varias veces, actualmente sólo se encuentra en uso la versión de QUIC desarrollada por Google. Sin embargo, la empresa de redes de distribución de contenido “Cloudflare”, tiene habilitada una página web (https://cloudflare-quic.com/) para llevar a cabo pruebas de la versión de QUIC del IETF.

5.7.1 CLIENTE FLUPKE

Para acceder al servidor de Cloudflare que soporta el protocolo QUIC del IETF, es necesario utilizar un cliente que utilice esta misma versión, pero actualmente no hay ningún navegador web con estas características. Por tanto, se ha procedido a utilizar un programa creado por Peter Doornbosch, el cual consiste en un cliente, llamado Flupke,

que utilizando la versión de QUIC del IETF envía una serie de paquetes al servidor que se especifique [30].

Para llevar a cabo esta prueba ha sido necesario utilizar el programa Virtual Box para crear una máquina virtual en el equipo. También ha sido necesario descargar e instalar en dicha máquina el sistema operativo Ubuntu 18.04 LTS. Después, se han descargado los programas necesarios: Wireshark, Java, un compilador de C, git…

Una vez creado el entorno de pruebas, se ha procedido a utilizar el cliente Flupke, para ello, el primer paso fue clonar el repositorio web con los archivos necesarios en la máquina virtual. El siguiente comando realiza esta acción:

git clone https://bitbucket.org/pjtr/flupke.git

Una vez clonado el repositorio, es necesario acceder a su ruta y clonar otros dos repositorios:

git clone https://bitbucket.org/pjtr/kwik.git

git clone https://bitbucket.org/pjtr/qpack.git

Después, hay que construir el fichero .jar con el siguiente comando

./gradlew build

Por último, se ejecuta el cliente comunicándose con el servidor de pruebas de QUIC de cloudflare:

java -cp build/libs/flupke.jar net.luminis.http3.Sample https://cloudflare-quic.com

Tras seguir estos pasos, se puede ver en la Figura 38 cómo en el terminal se indica que se han intercambiado 15 paquetes entre el cliente y el servidor.

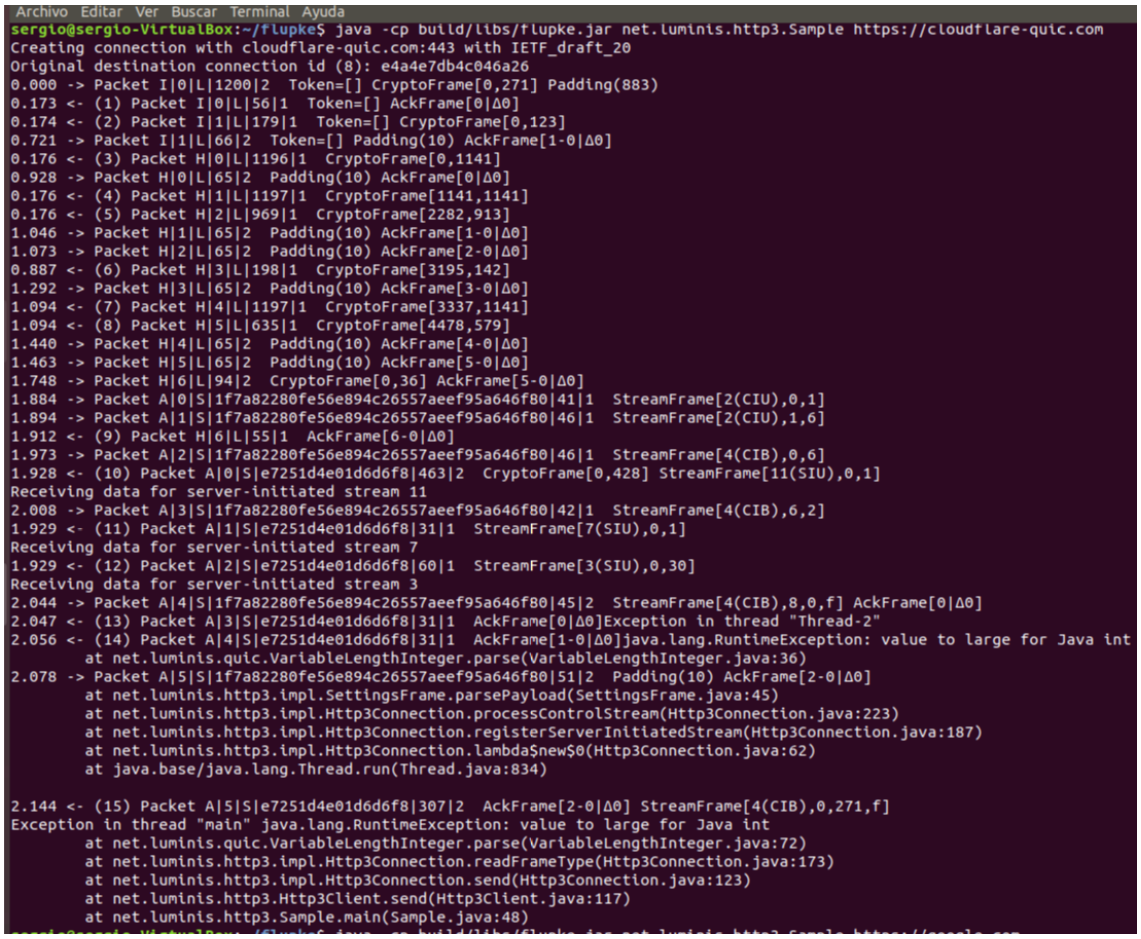
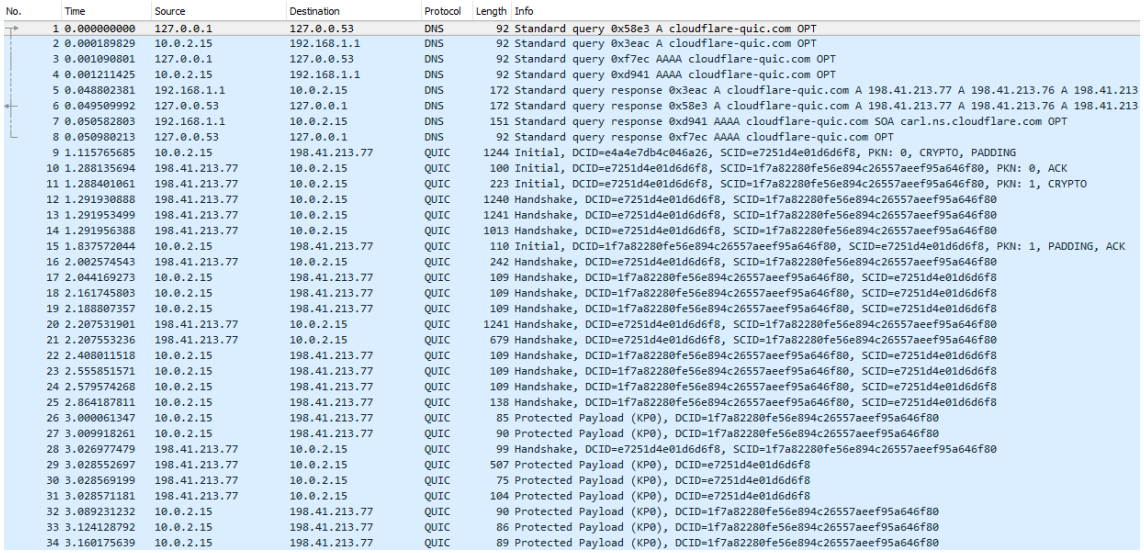


Figura 38: Comandos tras ejecutar Flupke.

Mientras se realizaba esta prueba, se ha capturado el tráfico con Wireshark y se ha comprobado cómo se genera un intercambio de paquetes UDP entre la máquina virtual y la IP 198.41.213.77 (la cual se ha comprobado con el servicio whois de, https://bandaancha.eu/whois , que pertenece a Cloudflare).

La versión de Wireshark que se estaba utilizando en la máquina virtual era la 2.6.8, que veía la comunicación UDP con el puerto 443 en el servidor, pero no era capaz de decodificar el contenido de los paquetes UDP. Entonces se probó a analizar la captura con una versión de Wireshark más moderna, la 3.0.1. En esta nueva versión de Wireshark, el contenido de los paquetes UDP sí se reconoce como paquetes QUIC (es importante notar que ahora se trata de QUIC y no GQUIC), como se muestra en la Figura 39.

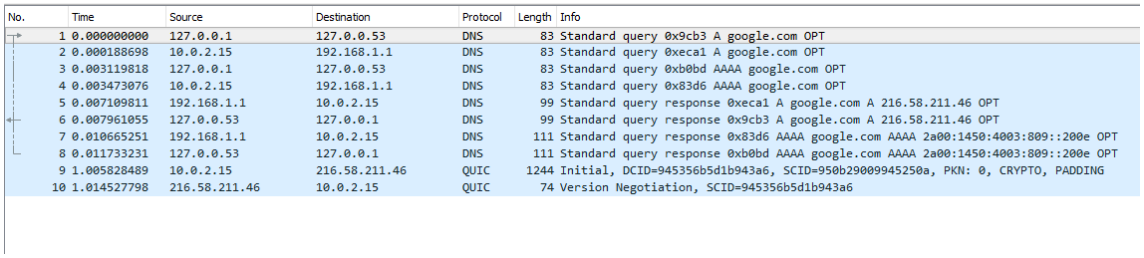


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.53	DNS	92	Standard query 0x58e3 A cloudflare-quic.com OPT
2	0.000189829	10.0.2.15	192.168.1.1	DNS	92	Standard query 0x3eac A cloudflare-quic.com OPT
3	0.001090801	127.0.0.1	127.0.0.53	DNS	92	Standard query 0xf7ec AAAA cloudflare-quic.com OPT
4	0.001211425	10.0.2.15	192.168.1.1	DNS	92	Standard query 0xd941 AAAA cloudflare-quic.com OPT
5	0.048802381	192.168.1.1	10.0.2.15	DNS	172	Standard query response 0x3eac A cloudflare-quic.com A 198.41.213.77 A 198.41.213.76 A 198.41.213
6	0.049509992	127.0.0.53	127.0.0.1	DNS	172	Standard query response 0x58e3 A cloudflare-quic.com A 198.41.213.77 A 198.41.213.76 A 198.41.213
7	0.050582803	192.168.1.1	10.0.2.15	DNS	151	Standard query response 0xd941 AAAA cloudflare-quic.com SOA carl.ns.cloudflare.com OPT
8	0.050980213	127.0.0.53	127.0.0.1	DNS	92	Standard query response 0xf7ec AAAA cloudflare-quic.com OPT
9	1.115765685	10.0.2.15	198.41.213.77	QUIC	1244	Initial, DCID=e4a4e7db4c046a26, SCID=e7251d4e01d6d6f8, PKN: 0, CRYPTO, PADDING
10	1.288135694	198.41.213.77	10.0.2.15	QUIC	100	Initial, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80, PKN: 0, ACK
11	1.288401061	198.41.213.77	10.0.2.15	QUIC	223	Initial, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80, PKN: 1, CRYPTO
12	1.291930888	198.41.213.77	10.0.2.15	QUIC	1240	Handshake, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80
13	1.291953499	198.41.213.77	10.0.2.15	QUIC	1241	Handshake, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80
14	1.291956388	198.41.213.77	10.0.2.15	QUIC	1013	Handshake, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80
15	1.837572044	10.0.2.15	198.41.213.77	QUIC	110	Initial, DCID=1f7a82280fe56e894c26557aeef95a646f80, SCID=e7251d4e01d6d6f8, PKN: 1, PADDING, ACK
16	2.002574543	198.41.213.77	10.0.2.15	QUIC	242	Handshake, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80
17	2.044169273	10.0.2.15	198.41.213.77	QUIC	109	Handshake, DCID=1f7a82280fe56e894c26557aeef95a646f80, SCID=e7251d4e01d6d6f8
18	2.161745803	10.0.2.15	198.41.213.77	QUIC	109	Handshake, DCID=1f7a82280fe56e894c26557aeef95a646f80, SCID=e7251d4e01d6d6f8
19	2.188807357	10.0.2.15	198.41.213.77	QUIC	109	Handshake, DCID=1f7a82280fe56e894c26557aeef95a646f80, SCID=e7251d4e01d6d6f8
20	2.207531901	198.41.213.77	10.0.2.15	QUIC	1241	Handshake, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80
21	2.207553236	198.41.213.77	10.0.2.15	QUIC	679	Handshake, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80
22	2.408011518	10.0.2.15	198.41.213.77	QUIC	109	Handshake, DCID=1f7a82280fe56e894c26557aeef95a646f80, SCID=e7251d4e01d6d6f8
23	2.555851571	10.0.2.15	198.41.213.77	QUIC	109	Handshake, DCID=1f7a82280fe56e894c26557aeef95a646f80, SCID=e7251d4e01d6d6f8
24	2.579574268	10.0.2.15	198.41.213.77	QUIC	109	Handshake, DCID=1f7a82280fe56e894c26557aeef95a646f80, SCID=e7251d4e01d6d6f8
25	2.864187811	10.0.2.15	198.41.213.77	QUIC	138	Handshake, DCID=1f7a82280fe56e894c26557aeef95a646f80, SCID=e7251d4e01d6d6f8
26	3.000061347	10.0.2.15	198.41.213.77	QUIC	85	Protected Payload (KP0), DCID=1f7a82280fe56e894c26557aeef95a646f80
27	3.009918261	10.0.2.15	198.41.213.77	QUIC	90	Protected Payload (KP0), DCID=1f7a82280fe56e894c26557aeef95a646f80
28	3.026977479	198.41.213.77	10.0.2.15	QUIC	99	Handshake, DCID=e7251d4e01d6d6f8, SCID=1f7a82280fe56e894c26557aeef95a646f80
29	3.028552697	198.41.213.77	10.0.2.15	QUIC	507	Protected Payload (KP0), DCID=e7251d4e01d6d6f8
30	3.028569199	198.41.213.77	10.0.2.15	QUIC	75	Protected Payload (KP0), DCID=e7251d4e01d6d6f8
31	3.028571181	198.41.213.77	10.0.2.15	QUIC	104	Protected Payload (KP0), DCID=e7251d4e01d6d6f8
32	3.089231232	10.0.2.15	198.41.213.77	QUIC	90	Protected Payload (KP0), DCID=1f7a82280fe56e894c26557aeef95a646f80
33	3.124128792	10.0.2.15	198.41.213.77	QUIC	86	Protected Payload (KP0), DCID=1f7a82280fe56e894c26557aeef95a646f80
34	3.160175639	10.0.2.15	198.41.213.77	QUIC	89	Protected Payload (KP0), DCID=1f7a82280fe56e894c26557aeef95a646f80

Figura 39: Conexión entre Flupke y servidor con la versión de QUIC del IETF.

Es decir, con esta prueba, se ha demostrado cómo es posible conectarse con un servidor de contenidos distribuido mediante la versión de QUIC del IETF, que aún no está estandarizada, con un cliente compatible con dicha versión, en este caso Flupke.

Siguiendo los mismos pasos, se ha establecido conexión entre el mismo cliente y google.com. En el terminal se indica que sólo se envía el primer paquete y el programa se para por un error de compatibilidad, ya que el servidor de Google no es compatible con la versión de QUIC del IETF. También se ha capturado el tráfico con Wireshark y, como era de esperar, se envía el primer paquete mediante QUIC, pero como Google no lo soporta responde con un mensaje solicitando la negociación de versión. Puede verse en la Figura 40.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.53	DNS	83	Standard query 0x9cb3 A google.com OPT
2	0.000188698	10.0.2.15	192.168.1.1	DNS	83	Standard query 0xeca1 A google.com OPT
3	0.003119818	127.0.0.1	127.0.0.53	DNS	83	Standard query 0xb0bd AAAA google.com OPT
4	0.003473076	10.0.2.15	192.168.1.1	DNS	83	Standard query 0x83d6 AAAA google.com OPT
5	0.007109811	192.168.1.1	10.0.2.15	DNS	99	Standard query response 0xeca1 A google.com A 216.58.211.46 OPT
6	0.007961055	127.0.0.53	127.0.0.1	DNS	99	Standard query response 0x9cb3 A google.com A 216.58.211.46 OPT
7	0.010665251	192.168.1.1	10.0.2.15	DNS	111	Standard query response 0x83d6 AAAA google.com AAAA 2a00:1450:4003:809::200e OPT
8	0.011733231	127.0.0.53	127.0.0.1	DNS	111	Standard query response 0xb0bd AAAA google.com AAAA 2a00:1450:4003:809::200e OPT
9	1.005828489	10.0.2.15	216.58.211.46	QUIC	1244	Initial, DCID=945356b5d1b943a6, SCID=950b29009945250a, PKN: 0, CRYPTO, PADDING
10	1.014527798	216.58.211.46	10.0.2.15	QUIC	74	Version Negotiation, SCID=945356b5d1b943a6

Figura 40: Tráfico entre Google y Flupke.

5.7.2 CLIENTE QUICHE

Muy similar al apartado anterior, se utiliza un programa llamado "quiche" con el que es posible lanzar un cliente y un servidor compatibles con la versión de QUIC del IETF [31].

Esta prueba también se ha realizado en una máquina virtual con sistema operativo Ubuntu 18.04 LTS. En primer lugar, es necesario instalar el programa rustup pero en una versión de desarrollo, no la versión estable. Para ello basta con introducir en un terminal el comando:

curl https://sh.rustup.rs -sSf | sh

Una vez comenzado el proceso de instalación, es necesario elegir una instalación personalizada para elegir como cadena de herramientas (toolchain) la opción nightly.

Para continuar es necesario que el terminal coja las variables de entorno necesarias, para ello basta con abrir un nuevo terminal o escribir este comando en el antiguo:

source .cargo/env

Una vez seguidos estos pasos, se procede con la instalación del programa quiche. Primero se clona el repositorio en el equipo, utilizando el comando:

git clonehttps://github.com/cloudflare/quiche.git

Tras clonar el entorno, se accede a la raíz donde se ha clonado quiche y se ejecutan los siguientes comandos para llevar a cabo la compilación:

git submodule update –init

cargo build --examples

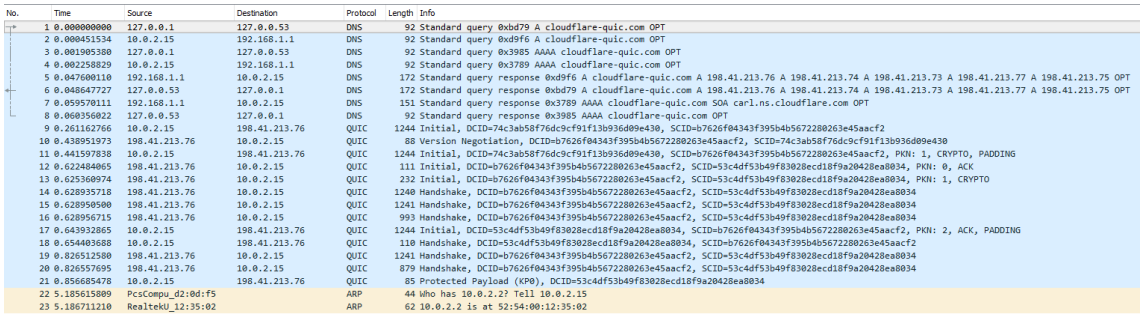
Una vez compilado, para probar que está todo bien se puede usar la siguiente línea:

cargo test

Ya está todo listo para lanzar el cliente y el servidor quiche. Primero se prueba a lanzar un cliente contra la página "cloudflare-quic.com" al igual que se hizo en el punto anterior con el cliente Flupke. Para ello basta con ejecutar el comando:

cargo run --example client -- --no-verify https://cloudflare-quic.com

Se ha utilizado Wireshark para capturar el tráfico generado al conectar este cliente con dicha página. En la Figura 41 se ve cómo esta conexión se hace mediante la versión QUIC del IETF.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.53	DNS	92	Standard query 0xbd79 A cloudflare-quic.com OPT
2	0.000451534	10.0.2.15	192.168.1.1	DNS	92	Standard query 0xd9f6 A cloudflare-quic.com OPT
3	0.001905380	127.0.0.1	127.0.0.53	DNS	92	Standard query 0x3985 AAAA cloudflare-quic.com OPT
4	0.002258829	10.0.2.15	192.168.1.1	DNS	92	Standard query 0x3789 AAAA cloudflare-quic.com OPT
5	0.047600110	192.168.1.1	10.0.2.15	DNS	172	Standard query response 0xd9f6 A cloudflare-quic.com A 198.41.213.76 A 198.41.213.74 A 198.41.213.73 A 198.41.213.77 A 198.41.213.75 OPT
6	0.048647727	127.0.0.53	127.0.0.1	DNS	172	Standard query response 0xbd79 A cloudflare-quic.com A 198.41.213.76 A 198.41.213.74 A 198.41.213.73 A 198.41.213.77 A 198.41.213.75 OPT
7	0.059570111	192.168.1.1	10.0.2.15	DNS	151	Standard query response 0x3789 AAAA cloudflare-quic.com SOA carl.ns.cloudflare.com OPT
8	0.060356022	127.0.0.53	127.0.0.1	DNS	92	Standard query response 0x3985 AAAA cloudflare-quic.com OPT
9	0.261162766	10.0.2.15	198.41.213.76	QUIC	1244	Initial, DCID=74c3ab58f76dc9cf91f13b936d09e430, SCID=b7626f04343f395b4b5672280263e45aacf2
10	0.438951973	198.41.213.76	10.0.2.15	QUIC	88	Version Negotiation, DCID=b7626f04343f395b4b5672280263e45aacf2, SCID=74c3ab58f76dc9cf91f13b936d09e430
11	0.441597838	10.0.2.15	198.41.213.76	QUIC	1244	Initial, DCID=74c3ab58f76dc9cf91f13b936d09e430, SCID=b7626f04343f395b4b5672280263e45aacf2, PKN: 1, CRYPTO, PADDING
12	0.622484065	198.41.213.76	10.0.2.15	QUIC	111	Initial, DCID=b7626f04343f395b4b5672280263e45aacf2, SCID=53c4df53b49f83028ecd18f9a20428ea8034, PKN: 0, ACK
13	0.625360974	198.41.213.76	10.0.2.15	QUIC	232	Initial, DCID=b7626f04343f395b4b5672280263e45aacf2, SCID=53c4df53b49f83028ecd18f9a20428ea8034, PKN: 1, CRYPTO
14	0.628935718	198.41.213.76	10.0.2.15	QUIC	1240	Handshake, DCID=b7626f04343f395b4b5672280263e45aacf2, SCID=53c4df53b49f83028ecd18f9a20428ea8034
15	0.628950500	198.41.213.76	10.0.2.15	QUIC	1241	Handshake, DCID=b7626f04343f395b4b5672280263e45aacf2, SCID=53c4df53b49f83028ecd18f9a20428ea8034
16	0.628956715	198.41.213.76	10.0.2.15	QUIC	993	Handshake, DCID=b7626f04343f395b4b5672280263e45aacf2, SCID=53c4df53b49f83028ecd18f9a20428ea8034
17	0.643932865	10.0.2.15	198.41.213.76	QUIC	1244	Initial, DCID=53c4df53b49f83028ecd18f9a20428ea8034, SCID=b7626f04343f395b4b5672280263e45aacf2, PKN: 2, ACK, PADDING
18	0.654403688	10.0.2.15	198.41.213.76	QUIC	110	Handshake, DCID=53c4df53b49f83028ecd18f9a20428ea8034, SCID=b7626f04343f395b4b5672280263e45aacf2
19	0.826512580	198.41.213.76	10.0.2.15	QUIC	1241	Handshake, DCID=b7626f04343f395b4b5672280263e45aacf2, SCID=53c4df53b49f83028ecd18f9a20428ea8034
20	0.826557695	198.41.213.76	10.0.2.15	QUIC	879	Handshake, DCID=b7626f04343f395b4b5672280263e45aacf2, SCID=53c4df53b49f83028ecd18f9a20428ea8034
21	0.856685478	10.0.2.15	198.41.213.76	QUIC	85	Protected Payload (KP0), DCID=53c4df53b49f83028ecd18f9a20428ea8034
22	5.185615809	PcsCompu_d2:0d:f5		ARP	44	Who has 10.0.2.2? Tell 10.0.2.15
23	5.186711210	RealtekU_12:35:02		ARP	62	10.0.2.2 is at 52:54:00:12:35:02

Figura 41: Cliente quiche con Cloudflare-quic.com

De la misma manera se ha lanzado el cliente quiche con otras páginas como google.com, akaquic.com, akamai.com y cloudflare.com. Sin embargo, como ocurría con Flupke, estas páginas no son compatibles con la versión de QUIC del IETF y por tanto puede verse una petición UDP entre la máquina y los servidores, pero no se obtiene respuesta.

5.7.3 CLIENTE Y SERVIDOR QUICHE

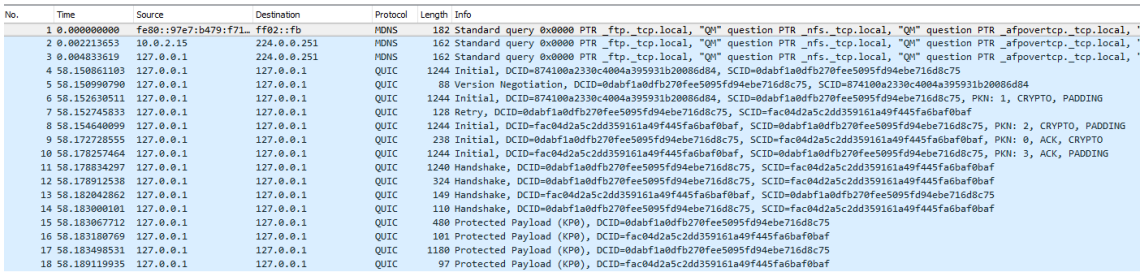
Por último, el programa quiche nos permite también lanzar un servidor compatible con la versión QUIC del IETF. Con el siguiente comando se lanza este servidor en localhost y la raíz del servidor es la ruta raíz desde donde se lanza el comando:

cargo run --example server -- --name localhost --root ./

Por último, para probar la conexión entre el cliente y el servidor quiche, se lanza un cliente contra ese mismo servidor en el puerto 4433 (que es el que usa por defecto):

cargo run --example client -- --no-verify https://localhost:4433/Cargo.toml

Se captura el tráfico generado con Wireshark y se muestra en la Figura 42. Se trata de tráfico QUIC del IETF. Por tanto, la prueba ha tenido éxito, es posible conectarse con un cliente a un servidor si ambos soportan esta versión del protocolo.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::97e7:b479:f71…	ff02::fb	MDNS	182	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.local, "QM" question PTR _afpovertcp._tcp.local, "
2	0.002213653	10.0.2.15	224.0.0.251	MDNS	162	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.local, "QM" question PTR _afpovertcp._tcp.local, "
3	0.004833619	127.0.0.1	224.0.0.251	MDNS	162	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.local, "QM" question PTR _afpovertcp._tcp.local, "
4	58.150861103	127.0.0.1	127.0.0.1	QUIC	1244	Initial, DCID=874100a2330c4004a395931b20086d84, SCID=0dabf1a0dfb270fee5095fd94ebe716d8c75
5	58.150990790	127.0.0.1	127.0.0.1	QUIC	88	Version Negotiation, DCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, SCID=874100a2330c4004a395931b20086d84
6	58.152630511	127.0.0.1	127.0.0.1	QUIC	1244	Initial, DCID=874100a2330c4004a395931b20086d84, SCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, PKN: 1, CRYPTO, PADDING
7	58.152745833	127.0.0.1	127.0.0.1	QUIC	128	Retry, DCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, SCID=fac04d2a5c2dd359161a49f445fa6baf0baf
8	58.154640099	127.0.0.1	127.0.0.1	QUIC	1244	Initial, DCID=fac04d2a5c2dd359161a49f445fa6baf0baf, SCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, PKN: 2, CRYPTO, PADDING
9	58.172728555	127.0.0.1	127.0.0.1	QUIC	238	Initial, DCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, SCID=fac04d2a5c2dd359161a49f445fa6baf0baf, PKN: 0, ACK, CRYPTO
10	58.178257464	127.0.0.1	127.0.0.1	QUIC	1244	Initial, DCID=fac04d2a5c2dd359161a49f445fa6baf0baf, SCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, PKN: 3, ACK, PADDING
11	58.178834297	127.0.0.1	127.0.0.1	QUIC	1240	Handshake, DCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, SCID=fac04d2a5c2dd359161a49f445fa6baf0baf
12	58.178912538	127.0.0.1	127.0.0.1	QUIC	324	Handshake, DCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, SCID=fac04d2a5c2dd359161a49f445fa6baf0baf
13	58.182042862	127.0.0.1	127.0.0.1	QUIC	149	Handshake, DCID=fac04d2a5c2dd359161a49f445fa6baf0baf, SCID=0dabf1a0dfb270fee5095fd94ebe716d8c75
14	58.183000101	127.0.0.1	127.0.0.1	QUIC	110	Handshake, DCID=0dabf1a0dfb270fee5095fd94ebe716d8c75, SCID=fac04d2a5c2dd359161a49f445fa6baf0baf
15	58.183067712	127.0.0.1	127.0.0.1	QUIC	480	Protected Payload (KP0), DCID=0dabf1a0dfb270fee5095fd94ebe716d8c75
16	58.183180769	127.0.0.1	127.0.0.1	QUIC	101	Protected Payload (KP0), DCID=fac04d2a5c2dd359161a49f445fa6baf0baf
17	58.183498531	127.0.0.1	127.0.0.1	QUIC	1180	Protected Payload (KP0), DCID=0dabf1a0dfb270fee5095fd94ebe716d8c75
18	58.189119935	127.0.0.1	127.0.0.1	QUIC	97	Protected Payload (KP0), DCID=fac04d2a5c2dd359161a49f445fa6baf0baf

Figura 42: Cliente quiche con servidor quiche.

5.8 CONCLUSIONES DEL CAPÍTULO

Actualmente en Internet, para servir contenidos web sólo se está utilizando la versión de QUIC desarrollada por Google. De hecho, ningún navegador web de uso extendido es compatible en su versión más reciente con la versión de QUIC del IETF. Sin embargo, la versión del IETF ya está siendo considerada por empresas de redes de distribución de contenido, como Akamai y Cloudflare, y por capturadores de tráfico como Wireshark. De hecho, Cloudflare tiene habilitada una página web que ya utiliza esta versión con clientes que la soporten, pero al no haber ningún navegador web que sea compatible con esta versión actualmente, de momento sólo se puede utilizar este protocolo con clientes como Flupke.

En cuanto a los software de servidores web como los mencionados en este capítulo, la mayoría no son compatibles aún con este protocolo, pero ya hay varias empresas que los han incorporado a sus servidores y se espera que muchas más empresas incorporen este protocolo una vez se estandarice.

En conclusión, el protocolo QUIC, se está utilizando a día de hoy en pocas páginas web, sin embargo, estas páginas representan una gran cantidad del tráfico de Internet. Se estima que actualmente QUIC representa el 7% del tráfico web global [21]. Se espera que esto cambie drásticamente cuando el IETF finalice la estandarización de su versión de QUIC, en ese momento aumentarán el número de navegadores web y servidores web que utilizarán esta tecnología, llegando a gestionarse la mayoría del tráfico web global mediante este protocolo.

6. ORGANIZACIÓN DEL PROYECTO

En este capítulo expone la planificación, el impacto socio-económico y el presupuesto del proyecto.

6.1 PLANIFICACIÓN DEL PROYECTO

La realización de este proyecto se divide en tres tareas principales con diferentes subtareas:

- Documentación: Esta fue la fase inicial. Comenzada en diciembre de 2018 y finalizada en febrero de 2019. Consta de las siguientes subtareas:
    - Lectura de diferentes drafts del IETF, RFCs, documentos científicos, conferencias… con el objetivo de entender el funcionamiento y las limitaciones de los distintos protocolos que constituyen Internet. 18 horas.
    - Lectura de diferentes drafts del IETF, conferencias, blogs y documentos científicos acerca del protocolo QUIC, con el fin de entender su funcionamiento y las ventajas e inconvenientes que pueda aportar. 35 horas
    - Elaboración de pequeños resúmenes y esquemas que recopilen la información más relevante obtenida durante la documentación. 20 horas.
- Pruebas: Esta fase consistió en llevar a cabo diferentes medidas y pruebas para evaluar el uso del protocolo QUIC en Internet:
    - Estudio de QUIC en Wireshark. 6 horas.
    - Estudio y pruebas de uso de las herramientas de Chrome para capturar registros de red y analizarlos. 11 horas.
    - Diseño de las primeras comprobaciones y pruebas sobre el funcionamiento del protocolo y enfrentamiento a problemas surgidos. 22 horas.
    - Instalación de diferentes navegadores web y estudio de su compatibilidad con QUIC. (Además, en el caso de que fueran compatibles, estudio de extensiones y configuraciones del navegador relacionadas con el protocolo). 13 horas.

- Instalación del entorno Linux y los programas necesarios para probar clientes y servidores de QUIC del IETF. Estudio del funcionamiento de los programas y diseño y realización de pruebas. 24 horas.
- Diseño y realización de experimentos para descubrir el soporte de QUIC en diferentes sitios web. 10 horas.
- Estudio del funcionamiento de QUIC mediante capturas de tráfico. 12 horas.
- Estudio y comprobación de la compatibilidad de QUIC con diferentes redes de distribución de contenido. 14 horas.
- Estudio y comprobación de la compatibilidad de QUIC con softwares de servidores web como Apache. 6 horas.

• <u>Redacción de la memoria:</u>

- Escritura de los capítulos a partir de la información recolectada en la fase de documentación. 96 horas.
- Revisión de los comentarios del tutor a lo largo de cada versión del borrador, corrección del documento, reestructuración del documento, recolección de nueva información y su escritura. 40 horas.

Actividad	Fecha inicio	Duración en horas	Fecha fin
Documentación esado del arte	17-dic	18	10-ene
Documentación protocolo QUIC	07-ene	35	15-feb
Elaboración de resúmenes y esquemas	17-dic	20	15-feb
Estudio de QUIC en Wireshark	18-feb	6	20-feb
Estudio herramientas de Chrome para guardar y analizar registros de red	21-feb	11	23-feb
Diseño de las primeras comprobaciones y primeras pruebas del protocolo	24-feb	22	29-feb
Estudio navegadores web	01-may	13	02-may
Pruebas con clientes y servidores QUIC del IETF	05-jun	24	08-jun
Pruebas soporte QUIC en diferentes servidores web	03-may	10	06-may
Estudio funcionamiento de QUIC	07-may	12	08-may
Estudio compatibilidad de QUIC con redes de distribución de contenido	08-may	14	25-may
Estudio de la compatibilidad de QUIC con softwares de servidores web	18-may	6	18-may
Escritura de la memoria	18-feb	96	10-jun
Revisión de la memoria y comentarios del tutor y correción del documento	20-abr	40	12-jun
Fecha inicio	17/12/2018		
Fecha fin	12/06/2019		

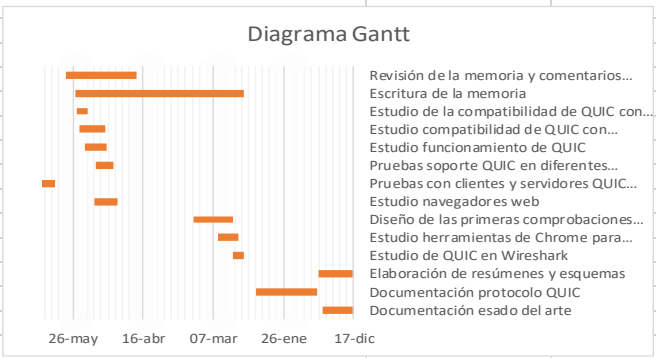


Figura 43: Gráfico Gantt

6.2 IMPACTO SOCIO-ECONÓMICO

En cuanto al impacto social, todos los usuarios de Internet podrán beneficiarse. De momento sólo se pueden beneficiar los usuarios de Google Chrome o Android que visiten servidores web que pertenezcan a Google o algunas páginas pequeñas (ya que es donde está implantado el protocolo). El beneficio no es sólo llevar a cabo conexiones más rápidas, lo cual es muy importante para clientes de streamings por ejemplo. También se beneficiarán todos los usuarios de conexiones más seguras, lo cual es muy importante hoy en día, ya que el número de ataques informáticos que se producen aumenta cada año.

Por otro lado, protocolos más eficientes se traducen en ahorro de consumo energético, lo cual es un importante impacto positivo tanto económico como social.

En el ámbito económico la sociedad no se verá directamente beneficiada. Sin embargo, Google, o las empresas que decidan sumarse al uso de este protocolo, se verán beneficiados económicamente, ya que el protocolo QUIC al permitir la multiplexación de streamings y reducir el número de RTT necesarios en las conexiones, es capaz de reducir el uso de recursos de CPU en los servidores. Por tanto, las empresas que utilicen este protocolo usarán sus servidores de forma más eficiente y podrán aumentar su número de usuarios.

El proyecto ha descrito QUIC, su uso actual y las herramientas disponibles para adoptarlo, y aunque tendría un valor económico para empresas y organizaciones que quisieran adoptar esta tecnología, el resultado del trabajo se deja con licencia “creative commons” para que pueda ser usado por quien lo necesite. Contribuyendo así a ayudar a extender el uso de tecnologías más eficientes.

6.3 PRESUPUESTO DEL PROYECTO

UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:

Sergio Lázaro Espinosa

2.- Departamento:

GITT

3.- Descripción del Proyecto:

- Titulo	**Evaluación del uso del protocolo QUIC en Internet**
- Duración (meses)	**8**
Tasa de costes Indirectos:	20%

4.- Presupuesto total del Proyecto (valores en Euros):

15.423,00 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (personas mes) [a]	Coste persona mes	Coste (Euro)	Firma de conformidad
Soto Campos Ignacio		Ingeniero Senior	0,4	4.289,54	1.715,82	
Lázaro Espinosa Sergio		Estudiante de Grado	2,4914	2.694,39	6.712,80	
					0,00	
					0,00	
		Hombres mes	**2,8914**	**Total**	8.428,62	

[a] 1 Persona mes = 131,25 horas. Máximo anual de dedicación de 12 personas mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 personas mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Programa "Wireshark"	0,00	80	2	60	0,00
Ordenador portátil Lenovo z50	499,00	80	8	60	53,23
Programa "Virtual Box"	0,00	50		60	0,00
sistema operativo Ubuntu 18.04 TLS	0,00	50		60	0,00
		100		60	0,00
					0,00
				Total	53,23

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \text{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	8.429
Amortización	53
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	1.696
Total	10.178

Figura 44: Presupuesto.

7. CONCLUSIÓN

En este trabajo se han visto las distintas limitaciones que tienen los protocolos más utilizados en los últimos años para gestionar tráfico web. Es importante destacar cómo, mediante descargas de contenido en paralelo, TCP persistente y HTTP/2 se ha reducido significativamente el tiempo necesario para establecer una conexión y llevar a cabo peticiones y respuestas al servidor. Sin embargo, estos protocolos tienen aún algunas limitaciones que han sido solucionadas con QUIC.

QUIC surgió como un protocolo de transporte desarrollado y utilizado por la empresa Google, y debido a las grandes ventajas que proporciona, el IETF ha desarrollado su propia versión y está en proceso de estandarizarla.

Se ha descrito en este trabajo el mecanismo utilizado por QUIC para establecer una conexión. Este handshake reduce el tiempo necesario para establecer una conexión segura y poder enviar paquetes al receptor, a 1RTT o 0RTT dependiendo del escenario. Esto supone milisegundos en la mayoría de los casos, sin embargo, en las comunicaciones que requieren de muchas conexiones como los streamings, esto supone una gran mejora para la latencia de la comunicación.

Otra gran mejora que surge con QUIC es la eliminación de problemas surgidos por “Head of Line Blocking”. Esto es posible gracias al sistema que utiliza QUIC de multiplicación de streams.

En cuanto a la seguridad, QUIC utiliza dos niveles, una primera conexión protegida por criptografía asimétrica y después criptografía simétrica, lo cual le proporciona la misma seguridad o más que los protocolos predecesores.

Se ha analizado el uso actual de QUIC en Internet y se ha averiguado que el protocolo de Google es compatible con 2 de los navegadores web más usados (Chrome y Opera), representando estos navegadores más de la mitad de los usuarios de Internet.

En cuanto a servidores web, se ha estudiado la compatibilidad de QUIC con las 15 páginas más visitadas en Internet, siendo sólo compatible este protocolo con 2 de ellas, pero estas dos son la primera y la segunda páginas más visitadas. Además, se ha comprobado que

QUIC ya es compatible con varias páginas web pequeñas que no son propiedad de Google. Por tanto, aunque este protocolo todavía no esté implementado en un gran número de páginas web, representa una parte significativa del tráfico web global.

Como se ha visto en este trabajo, el IETF y grandes empresas de distribución de contenidos como Akamai o Cloudflare han mostrado mucho interés en el protocolo. Este interés es comprensible debido a las ventajas que se han descrito en este trabajo, como la mejora de la latencia en las comunicaciones, reducción del uso de CPU en servidores, solución a limitaciones de los protocolos anteriores…

Tras las pruebas que se han realizado, se ha comprobado que ya hay implementaciones de clientes y servidores que pueden llevar a cabo una comunicación mediante la versión de QUIC del IETF, y que empresas como Cloudflare, ya tienen servidores compatibles con esta tecnología. Por tanto, se espera que cuando el IETF haya terminado la estandarización del protocolo, grandes empresas decidan implementarlo en sus servidores y clientes, pasando así, a ser QUIC el protocolo más utilizado para el tráfico web en un futuro no muy lejano.

En cuanto a las futuras líneas de trabajo en este campo, son muy amplias. Podría analizarse el uso del protocolo QUIC en dispositivos móviles. Actualmente no hay una versión del programa Wireshark para este tipo de dispositivos, ni ningún programa similar. Por tanto, una posible opción sería utilizar un router intermediario y desviar el tráfico del dispositivo móvil por ese router y analizar el tráfico que ha circulado por éste utilizando Wireshark.

Otra posible línea de estudio sería desarrollar un programa que pudiese explorar una lista de sitios web para ver si son compatibles con QUIC o no. En este trabajo se ha comprobado la compatibilidad manualmente con las 15 webs más utilizadas, con este programa podrían comprobarse rápidamente cientos o miles de páginas web.

También sería interesante llevar a cabo un estudio del porcentaje de tráfico que utiliza QUIC. En este trabajo se han mostrado datos de terceros, pero si se llega a un acuerdo con alguna universidad o un ISP, podrían recopilarse suficientes datos como para hacer una estimación directa de la cantidad de tráfico que lo utiliza. Por supuesto, habría que estudiar cómo recopilar esta información y garantizar la privacidad y anonimidad de los usuarios.

8. REFERENCIAS

[1] J. Roskind, “Experimenting with QUIC”, Chromium Blog, https://blog.chromium.org/2013/06/experimenting-with-quic.html, último acceso junio de 2019.

[2] Oracle, “Modelo de arquitectura del protocolo TCP/IP”, en *Guía de administración del sistema: servicios IP*, Redwood City, California, agosto 2011, 39-44, disponible en: https://docs.oracle.com/cd/E19957-01/820-2981/ipov-10/index.html.

[3] “Protocolo de transferencia de Hipertexto”, https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto, último acceso mayo 2019.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, y T. Berners-Lee, "Hypertext Transfer Protocol, HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, junio 1999, https://www.rfc-editor.org/info/rfc2616.

[5] J. Postel, “Transmission Control Protocol”, RFC 793, DOI 10.17487/RFC0793,

septiembre 1981, https://es.wikipedia.org/wiki/Segmento_TCP.

[6] “Reporte Anual de Ciberseguridad de Cisco 2018”, Cisco, https://www.cisco.com/c/dam/global/es_mx/solutions/pdf/reporte-anual-cisco-2018-espan.pdf.

[7] E. Rescorla, “HTTP Over TLS”, RFC 2818, DOI 10.17487/RFC2818, mayo 2000, https://www.rfc-editor.org/info/rfc2818.

[8] M. Bashyam, M. Jethanandani y A. Ramaiah, "TCP Sender Clarification for Persist Condition", RFC 6429, DOI 10.17487/RFC6429, diciembre 2011, https://www.rfc-editor.org/info/rfc6429.

[9] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, RFC 8446, DOI 10.17487/RFC8446, agosto 2018, https://www.rfc-editor.org/info/rfc8446.

[10] M. Belshe, R. Peon, y M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, mayo 2015, https://www.rfc-editor.org/info/rfc7540.

[11] J. Rüth, I. Poese, C. Dietzel y O. Hohlfeld, "A First Look at QUIC in the Wild", DOI: 10.1007/978-3-319-76481-8_19, marzo 2018, disponible en https://www.researchgate.net/publication/323483096_A_First_Look_at_QUIC_in_the_Wild.

[12] "Browser Market Share", https://www.netmarketshare.com/, ultimo acceso mayo 2019.

[13] Working Group QUIC del IETF, https://datatracker.ietf.org/wg/quic/charter/, ultimo acceso junio 2019.

[14] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)", IETF, Internet draft, draft-ietf-quic-http-20, 23 abril, 2019.

[15] M. Kuehlewind y B. Trammell, "Manageability of the QUIC Transport Protocol", IETF, Internet draft, draft-ietf-quic-manageability-04, 24 abril, 2019

[16] M. Thomson, "Version-Independent Properties of QUIC", IETF, Internet draft, draft-ietf-quic-invariants-3, 12 abril, 2019.

[17] J. Iyengar y M. Thomson, "A UDP-Based Multiplexed and Secure Transport", IETF, Internet draft, draft-ietf-quic-transport-19, 11 marzo, 2019.

[18] B. Trammell y M. Kuehlewind, "The QUIC Latency Spin Bit", Internet draft, draft-ietf-quic-spin-exp-01, Octubre 2018.

[19] A. Niroshan, "Understanding QUIC wire protocol", https://medium.com/@nirosh/understanding-quic-wire-protocol-d0ff97644de7, último acceso mayo 2019.

[20] S. Ohtsu, "Technical Overview of QUIC" presentada en HTTP/2 Conference Japan 2014, Japón, 3 noviembre 2014.

[21] A. Langley et al., "The QUIC Transport Protocol: Design and Internet-Scale Deployment" Proceeding of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM'17, Los Angeles, California, USA, 21-25 agosto 2017.

[22] T. Jager, J. Schwenk y J. Somorovsky, "On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption" presentada en 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA, 12-16 octubre 2015. [En línea]. Disponible en: https://dl.acm.org/citation.cfm?id=2813657.

[23] "QUIC in the wild", https://brave.com/quic-in-the-wild/ , ultimo acceso junio 2019.

[24] "Using modern Protocols like HTTP/2 and QUIC", https://blog.delouw.ch/2018/03/02/using-modern-protocols-http2-quic/ , último acceso mayo 2019.

[25] "Experience the LiteSpeed Difference", https://www.litespeedtech.com/products/litespeed-web-server/features/quic-support, ultimo acceso junio 2019.

[26] "Usage of QUIC for websites" https://w3techs.com/technologies/details/ce-quic/all/all, último acceso mayo 2019.

[27] "Red de distribución de contenido", https://www.akamai.com/es/es/resources/content-distribution-network.jsp, último acceso junio 2019.

[28] "QUIC Native Platform Support for Media Delivery Products", https://community.akamai.com/customers/s/article/FAQ-QUIC-Native-Platform-Support-for-Media-Delivery-Products?language=en_US, último acceso mayo 2019.

[29] "Get a head start with QUIC", https://blog.cloudflare.com/head-start-with-quic/, último acceso mayo 2019.

[30] P. Doornbosch, "Java HTTP3 Client", Flupke, https://bitbucket.org/pjtr/flupke/src/master/, último acceso junio 2019.

[31] "Savoury implementation of the QUIC protocol and HTTP/3", https://github.com/cloudflare/quiche, último acceso junio 2019.

ANEXO I

ABSTRACT

Nowadays, the Internet structure is based on a protocol stack. This stack consists of different levels containing varied protocols which gives support to the upper levels. The different levels on the TCP/IP stack are: application, transport, network, data link and physical. The most used system to manage web traffic is the protocol HTTP/2 (application level) over TLS and TCP (transport level).

HTTP is a protocol which manages requests and responses between a server and a client. This protocol needs for a security layer, as TLS, and a transport protocol, as TCP. Those requests and responses are the way to download contents from a web page and exchange data. Most of web pages have a lot of different contents such as advertisements, videos, images, texts… and these contents can be located in different servers.

At the beginning it was necessary to establish a single TCP connection and a single HTTP connection for downloading each content in a web page. TCP uses the mechanism known as three way handshake to carry out these connections, but that mechanism takes 1RTT before allowing the client to send a request to the server.

Currently, most of web pages have hundreds of different pieces of content, therefore, performing all the needed connections with this system would take hundreds of RTTs just for the multiple connections establishments. In addition, there are a lot of communications such as streamings which need fast connections and low latency. In order to reduce the number of connections appeared TCP persistent, which is able to maintain a unique TCP connection to download multiple pieces of content. Similar to this, HTTP evolved to HTTP/2 which a multiplexing system that let parallel content download and can manage multiple requests and responses on the same HTTP connection.

TLS also updated to a new version known as TLS 1.3 which increased the security and reduced the number of RTT needed in the handshake.

With the idea of carrying on with these improvements, QUIC emerges. A protocol developed by Google that replaces some of the layers in the TCP/IP protocol stack. In

terms of transport, QUIC is based on UDP instead of TCP and does not require of another security protocol such as TLS, since the protocol itself contains different measures to offer the same level of security.

QUIC emerges with the aim of reducing latency in HTTP communications over the Internet. TCP uses the mentioned three way handshake, however, QUIC use a different handshake system based on credentials that it has stored from a previous connections between those computers and eliminates the handshake packets that are redundant or unnecessary. Therefore, it allows connections to be established faster.

As mentioned before, QUIC was originally developed by Google, however, attracted by its advantages over TCP and HTTP/2, the IETF is working on their own version to standardize the protocol. Therefore IETF version is not in use yet. Nevertheless, Google version is already in use without being standardized. This is possible because Google owns both sides of the communication (client side if you use Google Chrome web browser and server side if you try to connect with Google servers such as Gmail, Youtube…). So Google decided to implement their protocol over the application level and use UDP for transport.

This is very important, since, if Google had not controlled both extremes, they would have to develop this protocol at the transport level, standardize it and allow the operating system developers to implement them in their systems. Another option would be to do it also at application level and try that the rest of companies in the market decide to add it to their browsers. (but this option is more complicated).

Both versions of the protocol provide the same characteristics (low latency, solution of the Head of Line Blocking problem…). Nevertheless, they have some design differences. Regarding to packet headers, IETF version has two different structures, the long header (used during handshake), and short header (used during the communication). On the opposite side, Google, use the same header for handshake and communication. Both versions include in their headers fields such as packet number, version (which indicates the protocol version they are using for the communication) and connection ID (which is a random value that identifies the connection and provide security).

Going on with the design of QUIC, the protocol includes a loss recovery mechanism similar to TCP, with the difference, that QUIC assigns a different number for each packet,

even if the original packet is missed, the forwarded packet will have a different packet number.

QUIC also counts with a flow control mechanism. In TCP, flow control mechanism is based on a window which is updated periodically, whose size determine the maximum bytes the server or client accepts for each delivery. QUIC uses two different Windows for flow control, flow window, analogous to TCP, and stream window, which determinate the maximum size a stream can have.

Not only flow control and loss recovery are important to carry out a fluent and stable communication, congestion avoidance is significant too. QUIC's congestion avoidance, such as TCP's, are based on cubic functions. First, the congestion window will increase rapidly until it reaches the first turning point, after reaching this size, the window will be some time between the two turning points trying to find balance and stabilization. Once it has stabilized, it grows rapidly again.

Other aspects of the design are the different handshakes QUIC performs depending on the scenario. If the client does not know the server from a previous connection, and does not have stored information about it on its cache memory, then the client will send an incomplete Client Hello message. As this packet is incomplete for the connection establishment, the client will send a Rejection message containing the necessary info to generate (server and client) the same symmetric cryptography key. Then the client will be able to send a Complete Client Hello and immediately after, it can send data. These systems takes for 1RTT. However, there is another scenario. When client and server had a recently previous connection and they have stored the necessary info to stablish a secure connection. In this case, the client will send directly a Complete Client Hello with the info it has stored, and it can send data immediately after, taking for 0RTT to stablish the connection previous to the data exchange. This handshake saves 2 RTT for connection establishment compared to HTTP/2 over TLS 1.3 over persist TCP. This may not mean much time since 2 RTT are usually milliseconds, however in streamings or on servers with hundreds of connections in a few time interval, this is a great time saving that allows to decrease the latency.

Another important aspect of QUIC is the use of streams. QUIC uses different streams to manage the download of each content. In this way, after the header of the packets, the separated information is sent in different streams that can have a variable size according

to the needs of the communication. A single stream can occupy the whole package if it is not necessary to transmit by other streams. However, the intention of QUIC is that the streams are small.

Once you have finished sending information by a stream, it is possible to cancel this stream without canceling the entire connection between the ends. This means that access to a large number of web pages or applications is faster. Currently web pages are not only composed of a text, but have images, videos, ads, and a large number of web elements.

With the arrival of this protocol, these web queries can be done by establishing a single connection and obtaining the necessary information for the different web applications by means of streams that can be canceled once they are no longer used. Making a single connection involves fewer handshakes and fewer delays per connection establishment and, therefore, you can access information faster.

Moreover, as QUIC uses streams that are multiplexed in a single connection, the loss of a packet only blocks the content of the streams corresponding to that packet. TCP has to deliver the bytes in order and this can produce Head Of Line Blocking. However, in QUIC, different contents are sent in parallel in different streams that do not have to arrive in order. And therefore this type of blocking is eliminated.

Regarding to security, QUIC grants two levels of security to your connections. The first is given when you send a first encrypted message assuming that the server will understand that message. In case the server does not understand it, it will respond with the possible algorithms that you can use to encrypt them. Thus, the client would send the encrypted data again with one of those algorithms that the server supports. The second level is given by an ephemeral key Diffie-Hellman.

During the handshake, a key exchange must be carried out that provides server authentication, optionally client authentication, different keys generated that are not related to other connections, and use of keys to encrypt the packets. Both in the case that the connection establishment requires 1 RTT and if it requires 0 RTT.

QUIC is protected from denial of service attack, STK reuse attack, Slowloris attack, stream fragmentation attack, version downgrade attack, Manger attack, Bleichenbacher attack…

In this document, the current use of QUIC on the Internet has been analyzed and it has been found that the Google protocol QUIC is compatible with 2 of the most used web browsers (Chrome and Opera), these browsers represent more than a half of Internet users. Other famous browsers such as Edge or Firefox do not support this protocol yet.

Regarding to web servers, the compatibility of QUIC with the 15 most visited web pages on the Internet has been studied, being only compatible with this protocol 2 of them, but these are the first and second most visited pages. In addition, it has been verified that QUIC is compatible with several small web pages that are not owned by Google. Therefore, although this protocol is not yet implemented in a large number of web pages, it represents a significant ammount of global web traffic.

As mentioned before, the IETF and some content distribution companies such as Akamai or Cloudflare have shown great interest in the protocol. This interest is understandable due to the advantages that have been described in this document, such as the improvement of the latency in communications, reduction of the use of the CPU in the servers, the solution to the limitations of the previous protocols ...

After the tests that have been carried out, it has been verified that there are already implementations of clients and servers that can entablish a communication through the IETF version of QUIC, and that companies such as Cloudflare, already have servers compatible with this technology. Therefore, it is expected that when the IETF has completed the standardization of the protocol, large companies decide to implement it on their servers and clients, thus becoming QUIC the most used protocol for web traffic in the near future.

In summary, the goal of QUIC is to reduce the number of packets exchanged in the handshake (improving the latency of communications), facilitate deployment or updates on the user side, preserve or increase the security that given by HTTP / 2 and improve limitations or eliminate problems from previous systems such as Head of Line Blocking.

Currently only Google’s QUIC version is being used in a few web pages, however, these pages represent a large amount of Internet traffic. It is estimated that currently QUIC represents 7% of global web traffic. Shortly, IETF will standardize his version of the protocol, and then most of the web traffic will be served by QUIC, and will increase the number of clients and servers who will be benefited by its use.